FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Valtteri Vuorio

EVALUATION OF LEARNING-BASED TECHNIQUES IN NOVEL VIEW SYNTHESIS

Master's Thesis
Degree Programme in Computer Science and Engineering
June 2021

Vuorio V. (2021) Evaluation of Learning-Based Techniques in Novel View Synthesis. University of Oulu, Degree Programme in Computer Science and Engineering, 62 p.

ABSTRACT

Novel view synthesis is a long-standing topic at the intersection of computer vision and computer graphics, where the fundamental goal is to synthesize an image from a novel viewpoint given a sparse set of reference images. The rapid development of deep learning has introduced a wide range of new ideas and methods in novel view synthesis where parts of the synthesis process are considered as a supervised learning problem. Specifically, neural scene representations paired with volume rendering have achieved state of the art results in novel view synthesis, but still remains a nascent field facing a lack of literature.

This thesis presents an overview of learning-based view synthesis, experiments with state-of-the-art view synthesis methods, evaluates them quantitatively and qualitatively and finally discusses their properties. Furthermore, we introduce a novel multi-view stereo dataset captured with a hand-held camera and demonstrate the process of collecting and preparing multi-view stereo datasets for view synthesis.

The findings in this thesis indicate that learning-based view synthesis methods excel at synthesizing plausible views from challenging scenes, including situations with complex geometry as well as transparent and reflective materials. Furthermore, we found that it is possible to render such scenes in real-time and greatly reduce the time to prepare a scene for view synthesis by using a pre-trained network that aggregates information from nearby views.

Keywords: deep learning, image-based rendering, machine learning, neural rendering, computer vision, computer graphics

Vuorio V. (2021) Koneoppimisen soveltaminen uuden näkymän synteesissä. Oulun yliopisto, Tietotekniikan tutkinto-ohjelma, 62 s.

TIIVISTELMÄ

Uuden näkymän synteesi on pitkäaikainen aihe konenäön ja tietokonegrafiikan risteyksessä, jossa tavoitteena on syntetisoida kuva uudesta näkökulmasta annetun kuvajoukon perusteella. Syväoppimisen nopea kehitys on synnyttänyt laajan kirjon uusia ideoita ja menetelmiä uuden näkymän synteesissä, jossa osia synteesiprosessista pidetään valvottuna oppimisongelmana. Erityisesti neuraaliset tilaesitykset yhdistettynä tilavuusrenderointiin ovat saavuttaneet huippuluokan tuloksia uuden näkymän synteesissä, mutta aihe on vielä kehittyvä tieteenala.

Tässä opinnäytetyössä esitetään yleiskatsaus oppimispohjaiseen näkymän synteesiin, suoritetaan kokeellista tutkimusta uusimmilla synteesimenetelmillä, arvioidaan niitä kvantitatiivisesti ja kvalitatiivisesti sekä lopuksi keskustellaan niiden ominaisuuksista. Lisäksi esitellään uusi stereokuvien muodostama tietoainesto ja esitetään prosessi, jolla kerätään ja valmistellaan kyseisiä tietoaineistoja näkymän synteesiä varten.

Työssä havaitaan, että oppimispohjaiset näkymäsynteesimenetelmät piirtävät erittäin aidolta näyttäviä näkymiä tietoaineiston pohjalta jopa tilanteissa, missä esiintyy monimutkaista geometriaa sekä läpinäkyviä ja heijastavia materiaaleja. Lisäksi havaitsimme, että syntetisointi on mahdollista suorittaa reaaliajassa ja että syntetisoinnin valmisteluaikaa voidaan myös lyhentää käyttämällä ennalta koulutettua verkkoa, joka kokoaa tietoja läheisistä näkymistä.

Avainsanat: syväoppiminen, kuvapohjainen renderointi, koneoppiminen, neuraalinen renderointi, konenäkö, tietokonegrafiikka

TABLE OF CONTENTS

FOREWORD

This thesis was conducted between March 2021 to June 2021 at the Center for Machine Vision and Signal Analysis (CMVS) in University of Oulu. I want to thank professor Janne Heikkilä for the opportunity to work on this exciting topic and for the helpful comments and feedback during the writing of this thesis. I would also like to thank PhD researcher Phong Nguyen for his technical supervision, insights and guidance during this work.

Oulu, June 5th, 2021

Valtteri Vuorio

LIST OF ABBREVIATIONS

3D	Three Dimensional Space
CNN	Convolutional Neural Network
DNN	Deep Neural Network
IBR	Image-based Rendering
LPIPS	Learned Perceptual Image Patch Similarity
MLP	Multi-layer Perceptron
MPI	Multi-plane Image
MSE	Mean-squared Error
MVS	Multi-view Stereo
NVS	Novel View Synthesis
PSNR	Peak-to-signal Noise Ratio
RGB	Red, Green, Blue
ReLU	Rectified Linear Unit
SfM	Structure from Motion
SSIM	Structural Similarity Index

1. INTRODUCTION

Novel view synthesis (NVS) is a long-standing topic at the intersection of computer vision and computer graphics, where the fundamental goal is to synthesize an image from a novel viewpoint given a sparse set of reference images. View synthesis algorithms use information from co-located images to render photo-realistic images from arbitrary viewpoints, enabling applications, e.g., in virtual navigation, telepresence or product visualization. Furthermore, recent work has demonstrated the possibility of extending view synthesis algorithms to many of the characteristics of a traditional computer graphics pipeline, including relighting, scene composition and real-time rendering. The development and adoption of view synthesis algorithms could have far reaching implications in society, as it enables a way to automate the process of modeling and rendering digital replicas of real-world environments, objects and human beings.

Many of the recent advancements in novel view synthesis can be attributed to the on-going revolution in deep learning. Deep learning is a methodology that aims to solve computational problems by training multi-layer networks to automatically learn feature representations from vast amounts of data and make informed decisions on their own. The deep learning revolution is a consequence of increasing amounts of available data and the advent of sophisticated deep learning frameworks that leverage automatic differentiation and modern graphics cards to let end users design and optimize increasingly complex networks. Deep learning has been shown to be effective for solving complex and ill-posed tasks when large amounts of supervised input-output data is available. This fits the goal of novel view synthesis quite well, as the problem is ill-posed and a set of viewpoint-image pairs can be considered as supervised data.

The rapid development of deep learning paired with the long tradition of work in classical computer vision has introduced a wide range of new ideas and methods in novel view synthesis where parts of the synthesis process are considered as a supervised learning problem and replaced with deep neural networks. However, there is a general lack of literature that reviews this new field as a whole and it is far from obvious what kind of network architecture and scene representation is most suitable for a particular use-case. We aim to answer that question by reviewing the field as a whole and by introducing a novel dataset that is used to conduct quantitative and qualitative analysis on a wide range of methods. Furthermore, we aim to demonstrate from first principles how to replicate the results with data captured from a hand-held camera.

1.1. Objective of the Thesis

The first objective of this thesis is to provide an overview of state of the art methods in learning-based view synthesis. We categorize the lines of work based on their underlying scene representation and provide a concise summary of the underlying concepts. After reviewing the state of the art in learning-based view synthesis we choose and review a selected number of open-source methods for further evaluation.

The second objective is to collect and prepare a novel multi-view stereo dataset for the purposes of evaluating the selected open-source methods, demonstrating best practices for collecting such datasets with a hand-held camera.

Finally, the third objective is to evaluate the selected open-source methods by conducting a quantitative and qualitative comparison and by experimenting with their properties using the collected dataset. We discuss in detail how the selected methods relate to each other, identify the context where each one is most applicable, as well as consider their potential applications and future research directions.

1.2. Structure of the Thesis

This thesis consists of seven chapters:

- Chapter 2 introduces the concept of novel view synthesis, reviews how the problem has been approached historically, and identifies some limitations of traditional view synthesis to establish the motivation behind learning-based techniques.
- Chapter 3 continues on the topic of novel view synthesis by first introducing the concept of deep learning and then reviewing literature that adopts deep learning and deep neural networks as part of the view synthesis process.
- Chapter 4 covers the data collection part of this thesis, demonstrating the practice of collecting and preparing a multi-view stereo dataset for the purposes of this thesis and presents the collected dataset.
- Chapter 5 describes the experimental setting in which we evaluate state of the art methods in learning-based novel view synthesis.
- Chapter 6 evaluates and compares the results of the conducted experiments both quantitatively and qualitatively.
- Chapter 7 discusses the results, implications and limitations of the conducted experiments and considers future research directions in learning-based novel view synthesis.
- Finally, the main contributions of the thesis are summarized in Chapter 8.

2. NOVEL VIEW SYNTHESIS

This chapter introduces the concept of novel view synthesis and reviews how the problem has been approached historically. Finally, we discuss some limitations of traditional view synthesis to establish the motivation behind learning-based techniques.

2.1. Problem Definition

The fundamental goal of novel view synthesis is to render an image from a novel viewpoint given a sparse set of reference images and corresponding camera poses, illustrated in Figure 1. As opposed to traditional computer graphics where the scene is constructed from hand-crafted 3D models, the goal of view synthesis algorithms is to use images captured from the real world as a medium for rendering scenes. The idea of using images as a rendering medium has remained a challenging and long-standing topic in academia due to a number of challenges. Some of the key challenges include camera pose estimation, inferring the geometric structure of the scene, modeling view-dependent lightning and gracefully handling missing information. In addition to synthesizing plausible views, the practicality of the algorithm imposes additional challenges, such as achieving fast rendering speed and low memory footprint.

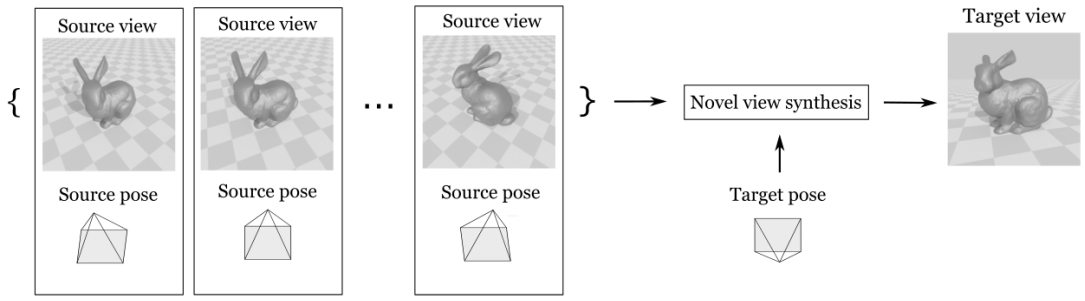


Figure 1. The goal of view synthesis is to render a target view given a set of source views and corresponding camera poses.

2.2. Traditional View Synthesis

The ways that novel view synthesis has been approached historically can be categorized to two different approaches, light field rendering and image-based rendering. The first, light field rendering, is based on directly interpolating between densely sampled images and the second, image-based rendering (IBR), takes advantage of geometrical information derived from multi-view stereo to synthesize views.

2.2.1. Light Field Rendering

Early work in novel view synthesis [1] acknowledges that the novel view synthesis problem can be reframed as approximating a light field representation, which is a continuous 5-dimensional function that returns incoming radiance L originating from the scene for a given position $\mathbf{x}$ and view direction $\mathbf{d}$:

$$L = L(\mathbf{x}, \mathbf{d}) \tag{1}$$

The light field captures the flow of light in the scene, making it possible to render novel views by sampling it from desired locations. Given a collection of source views and camera poses that densely cover the parameter space, the light field can be discretized into a regular grid and interpolated between samples to synthesize novel views [2] [3]. This discretization can be further reduced to a 4-dimensional grid by noting that radiance stays constant along a ray in free space. This approach was termed as the lumigraph [3], in which the light field of an enclosed scene is discretized using bounding planes, illustrated in Figure 2. However, this approach works only with a sufficiently dense collection of images. Chai et. al. [4] quantified this requirement by showing that light field rendering has a fundamental minimum sampling rate.

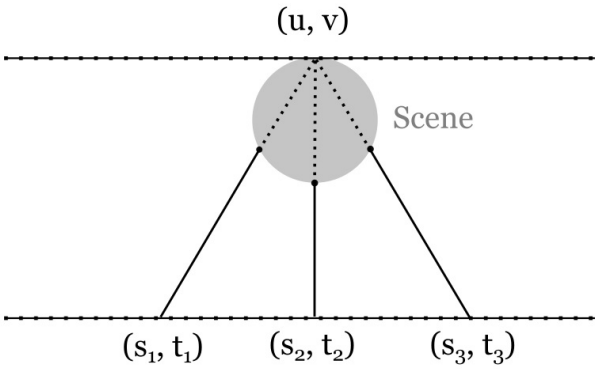


Figure 2. 2D-projection of the 4D-lumigraph. Any position and direction $(\mathbf{x}, \mathbf{d})$ within the scene is associated with a ray from (s, t) to (u, v). The light field is stored in discrete form in the two planes.

2.2.2. Image-Based Rendering

Image-based rendering refers to novel view synthesis techniques that utilize a geometrical estimate of the scene along with nearby views to synthesize novel views. The development of structure from motion (SfM) and multi-view stereo (MVS) algorithms introduced a reliable way to estimate the camera poses and geometry of the scene from a set of images, which became a foundational technique for image-based rendering. The goal of SfM is to find the parameters of each camera and the 3D coordinates of shared points in the scene. The traditional way of solving this problem involves an image correspondence search and an incremental reconstruction procedure, explained in more detail, e.g., in [5]. Multi-view stereo builds upon this technique by

associating each pixel of the source views with a globally consistent depth value [6]. The end result is a dense point cloud that can be further refined into a textured mesh using e.g. Delaunay triangulation [7].

Traditional IBR algorithms utilize MVS to estimate the surface geometry of the scene and use it to reproject nearby images into a novel viewpoint [8] [9]. As illustrated in Figure 3, the reprojection process involves projecting the pixels of nearby images to the surface geometry and back to the target view. The reprojected pixels can possibly overlap and end up in sub-pixel locations, which is handled by aligning the pixels with the target view by means of, e.g., inverse bilinear interpolation and then using a blending operation to form the final image.

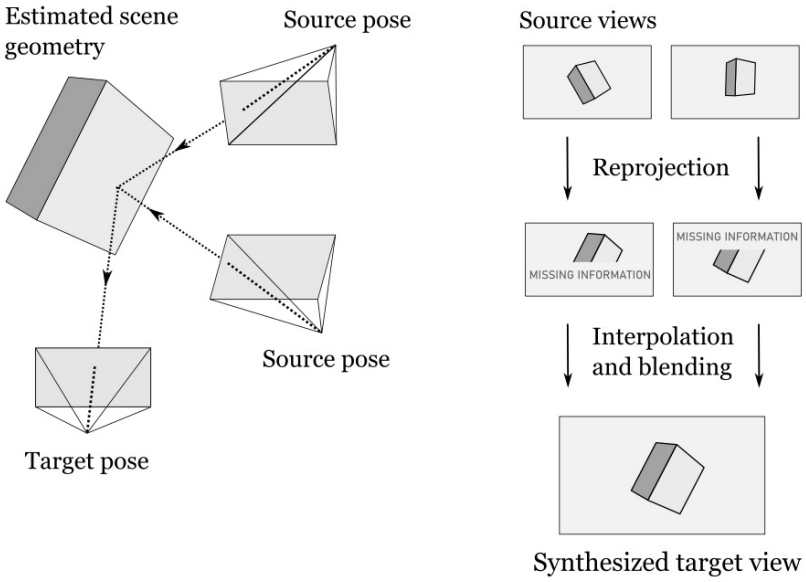


Figure 3. An overview of the process behind traditional image-based rendering.

The blending operation varies between authors, for example, Chaurasia et. al. [8] specifies the weights by camera orientation and the reliability of depth information at each pixel. Later works improve the geometry estimation with various modifications to MVS such as per-view meshes [10] and modeling depth uncertainty [11].

2.3. Limitations of Traditional View Synthesis

While the two discussed approaches can synthesize plausible views, they are limited in various aspects and have room for improvement. The approach of light field interpolation is only feasible when working with a sufficiently dense collection of images, as pointed out by Chai et. al [4]. Given the required sampling rate and the the high memory cost of discretizing the parameter space, light field rendering does not lend itself to large-scale scenes.

Image-based rendering is more scalable and can be integrated to existing graphics engines, however it struggles with it's own set of issues. Firstly, it is fundamentally limited by the accuracy of the reconstructed surface geometry. Scenes involving

detailed geometry and view-dependent materials are especially challenging for SfM and MVS to reconstruct, producing noisy or spurious geometry as well as missing regions. The blending process is at best a crude approximation to the various ways light can scatter from a surface, especially in scenarios with challenging light interactions such as reflection and refraction.

In summary, synthesizing views from scenes involving sparse views, detailed geometry or view-dependent materials remains a challenging task for traditional view synthesis and is one of the reasons why deep learning was later explored as a way to address the above mentioned issues.

3. LEARNING-BASED VIEW SYNTHESIS

This chapter continues on the topic of novel view synthesis by first introducing the concept of deep learning and then moving on to view synthesis methodologies that utilize deep neural networks as part of the view synthesis process. We categorize such methodologies into three lines of work, namely, image-based rendering, volumetric representations and neural scene representations.

3.1. Deep Learning

Deep learning is a subset of machine learning that aims to solve computational problems by training multi-layer networks to automatically learn feature representations from vast amounts of data and make informed decisions on their own [12]. The common way to approach a problem with deep learning is to reframe it as a supervised learning problem. Given a dataset of input-output pairs $(\mathbf{x}_i, \mathbf{y}_i)$, supervised learning aims to find a computational model $\mathbf{F}_{\boldsymbol{\theta}}(\mathbf{x})$ that predicts $\mathbf{y}$ for any given $\mathbf{x}$. The success of the model $\mathbf{F}_{\boldsymbol{\theta}}$ is measured with an appropriate loss function $\mathcal{L}$, such that the problem is considered solved when $\mathcal{L}$ is minimal with respect to the model parameters $\boldsymbol{\theta}$:

$$\boldsymbol{\theta} = \arg\min_{\theta} \sum_{i} \mathcal{L}(\mathbf{F}_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i) \tag{2}$$

Next, we look at deep neural networks as the choice of $\mathbf{F}_{\boldsymbol{\theta}}$ and how the above expression is minimized using first-order optimization techniques.

3.1.1. Deep Neural Networks

Deep neural networks (DNN) refer to models that are constructed by compositing several non-linear functions known as layers. Most network architectures define the layer as an affine transformation $\mathbf{W}\mathbf{x} + \mathbf{b}$ followed by a non-linear scalar function σ, also known as the activation function, applied element-wise. In this case, the parameters $\boldsymbol{\theta}$ are stored in $\mathbf{W}$ and $\mathbf{b}$, which are known as the weights and bias of the layer.

$$\mathbf{f}(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{3}$$

Assuming that N layers are applied in series, we arrive at what is called a feed-forward neural network:

$$\mathbf{F}(\mathbf{x}) = \mathbf{f}_1 \circ \cdots \circ \mathbf{f}_N(\mathbf{x}) \tag{4}$$

In practice, deep neural networks are not restricted to the feed-forward model and can have arbitrary inter-connections in the form of a directed graph, which consists of input variables, parameters and differentiable operations. An example of this is demonstrated in Figure 4.

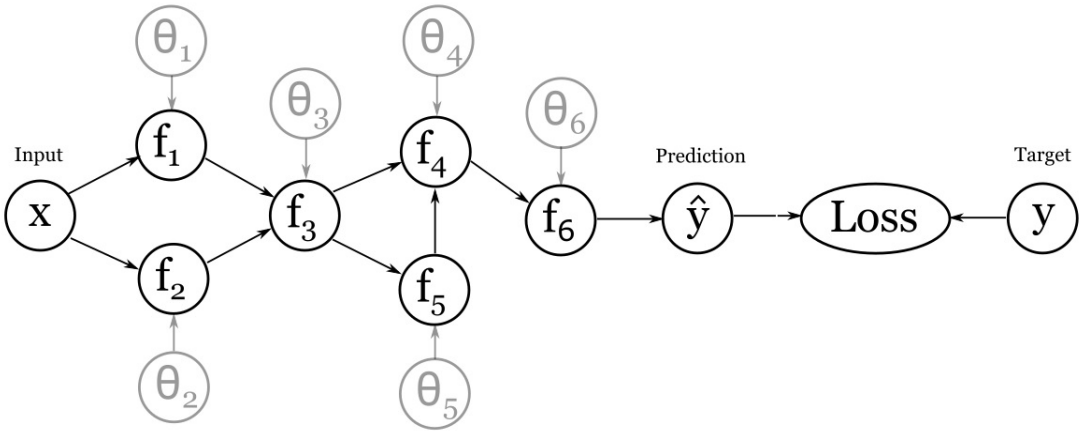


Figure 4. A deep neural network can be viewed as a directed graph, which consists of input variables, parameters $\boldsymbol{\theta}_i$ and differentiable operations $\boldsymbol{f}_i$.

3.1.2. Training DNNs

Training a DNN is the process of finding the parameters of the network that minimize the loss function as in Equation 2. After initializing the parameters of the network, the process begins by iteratively evaluating the network and loss function with training samples and updating the parameters with a first-order optimization technique. First-order optimization relies on the fact that for a given parameter $\boldsymbol{\theta}_i$, the gradient of the loss function with respect to said parameter points outwards from a local minimum. In other words, updating the parameter in the opposite direction should move towards a local minimum, which is adopted as the strategy to find the optimal parameters:

$$\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i - \alpha \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_i} \tag{5}$$

This update step is repeated for a number of times using a suitable learning rate α, until the loss reaches a sufficiently small value. The process is also known as gradient descent, which has been later improved with more advanced schemes such as AdaGrad [13] and ADAM [14].

Computing the gradient itself can be done with finite difference approximation, however this is quite inefficient as it requires a network evaluation for each perturbation. Instead, we can exploit the fact that each individual operation is differentiable and use the chain rule from calculus to evaluate the gradients more efficiently. To clarify the meaning of derivative for multi-dimensional functions, we define it as a matrix of component-wise elementary derivatives:

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_1}{\partial x_n} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix} \tag{6}$$

The above matrix is also known as the Jacobian matrix. Applying the chain rule from calculus, we arrive at the following expression for the gradient of the loss function with respect to a given parameter:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_i} = \frac{\partial \mathbf{f}_i}{\partial \boldsymbol{\theta}_i} \frac{\partial \mathcal{L}}{\partial \mathbf{f}_i} = \frac{\partial \mathbf{f}_i}{\partial \boldsymbol{\theta}_i} \Big[\sum_{j}^{\text{child nodes}} \frac{\partial \mathbf{f}_j}{\partial \mathbf{f}_i} \frac{\partial \mathcal{L}}{\partial \mathbf{f}_j} \Big] \tag{7}$$

In order to evaluate this expression, we start from the loss node and propagate backwards to each parameter in the computational graph, computing the Jacobian of each operation with respect to it's input along the way. This process is commonly known as backpropagation [15].

3.1.3. Evaluating DNNs

Measuring the success of a trained model $\mathbf{F}_{\boldsymbol{\theta}}$ using the training set $(\mathbf{x}_i, \mathbf{y}_i)$ can lead to misleading results if the network happens to learn the data points, i.e. $\mathbf{F}_{\boldsymbol{\theta}}(\mathbf{x}_i) \approx \mathbf{y}_i$, but fails to interpolate between values. An unbiased way to evaluate the model is to leave out a proportion of the samples from the training set and specifically use them to evaluate the success of the model. In novel view synthesis, this translates to leaving out a specific number of images from the training process and comparing them against the corresponding synthesized views using a suitable metric.

3.1.4. Common DNN Architectures

The DNN architectures that are relevant for this thesis are multi-layer perceptrons (MLP), convolutional neural networks (CNN) and sequence models.

Multi-layer perceptrons: MLPs [16] are characterized by a series of layers where the weight matrix is fully connected, i.e. each element in the matrix are non-zero and trainable. Because a fully connected weight matrix is computationally demanding, these networks typically operate on low-dimensional data. Historically, MLPs were used for classification tasks and are recently used to represent continuous signals [17], including continuous scene representations for novel view synthesis [18].

Convolutional neural networks: CNNs [19] were originally designed for image classification tasks and are characterized by a sparse weight matrix that operates within local image patches. Advanced CNNs such as ResNet [20] and U-Net [21] incorporate skip connections and batch normalization to improve convergence. In addition to classification, CNNs have seen a variety of uses in other domains, such as depth estimation, feature extraction and generative models.

Sequence models: Sequence models are networks that are designed to operate on variable length sequences. In order to reason about variable length input, sequence models adopt mechanisms such as feedback connections (e.g. long-short term memory network (LSTM) [22], gated recurrence unit (GRU) [23]) and multi-head attention (e.g. Transformer [24]). Sequence models have seen use in a wide range of domains, including natural language processing, machine translation and recently in novel view synthesis [25] [26].

3.2. Image-Based Rendering

In this section, we begin looking at novel view synthesis methodologies that utilize deep learning and deep neural networks by starting from approaches that have common qualities with traditional image-based rendering, discussed in Section 2.2.2.

3.2.1. Neural Image-Based Rendering

Neural image-based rendering strives to improve many of the steps taken in classical image-based rendering by augmenting or replacing parts of the view synthesis with supervised learning models. The first of these methods use classical 3D reconstruction to estimate the geometry of the scene and incorporate CNN-architectures to improve the blending process for the reprojected source views [27] [28] [29]. Specifically, Hedman et. al. [27] proposed to use a U-Net type CNN architecture to predict blending weights. An overview of this process is shown in Figure 5. Riegler et. al. [28] proposed a similar scheme where the source views are sent to a CNN encoder to extract image features, which are then fed to a GRU-based recurrent network for predicting blending weights. Finally, Thies et. al. [29] improved on rendering view-dependent effects by separately synthesizing diffuse and view-dependent components.

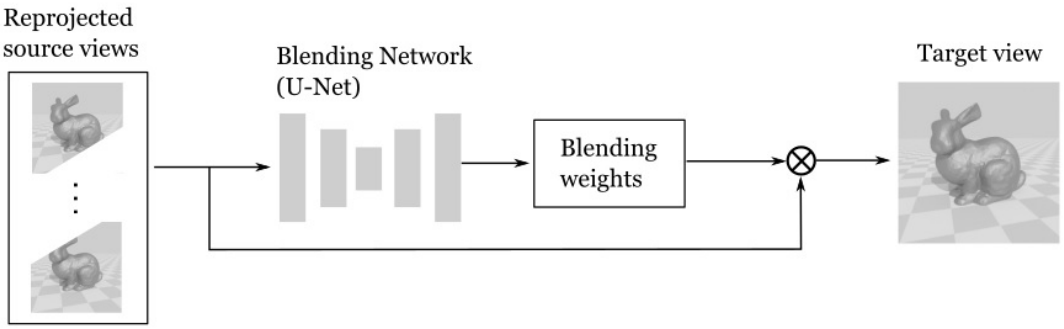


Figure 5. Overview of view synthesis using a blending network. The reprojected source views are sent to a blending network which predicts blending weights. The image is synthesized by adding together the weighted colors of each reprojected source view.

The above discussed methods successfully improve the blending process, removes the need for hand-crafted heuristics and can degrade more gracefully when faced with a lack of information. However, these are still limited to the accuracy of the 3D reconstruction and suffer from flickering when moving around the scene because the networks are not trained for temporal consistency.

Later works experiment with alternatives to classical 3D reconstruction by incorporating geometry estimation into the network architecture [30][31]. Choi et. al. [30] and Nguyen et. al. [31] predict a depth probability volume from each source view which are then warped into the target view by means of a homography. Finally, most recent work predicts density instead of depth and synthesizes the view via volume rendering [26].

3.2.2. Neural Re-Rendering

Neural re-rendering is a relatively short line of work that is similar to neural image-based rendering and was inspired by recent work in image-to-image translation [32]. Instead of directly using information from source images these methods opt to use a textured mesh [33] or a colored point cloud [34] [35] obtained from classical 3D reconstruction. These methods render the proxy geometry, i.e. point cloud or mesh, into graphics buffers using a traditional rasterizer and then send the result to a convolutional neural network to synthesize the final image. The purpose of the re-rendering network is to learn a mapping from graphics buffers such as color and depth into a plausible view. An image translation CNN [32] can be trained to perform this task, however later works show an improvement by equipping the proxy geometry with trainable latent codes to let the network learn additional features such as view-dependent lighting [33] [35]. An overview of this process is shown in Figure 6. Similar to early methods in neural image-based rendering, these methods are limited to the accuracy of the 3D reconstruction and suffer from flickering when moving around the scene. In addition, the re-rendering network needs to be trained for each scene separately.

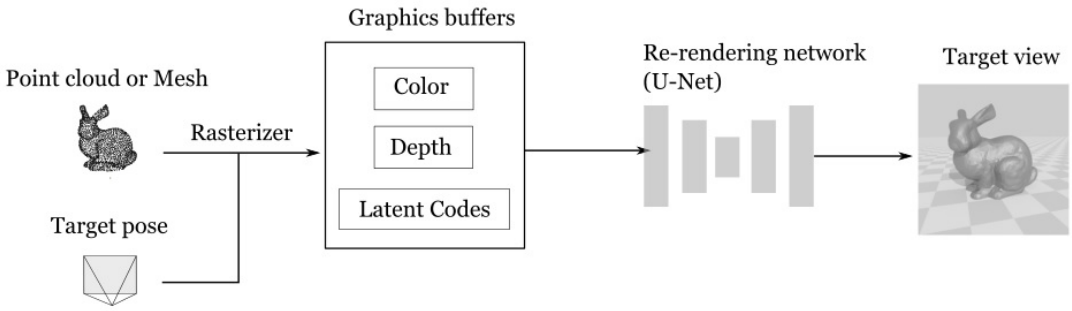


Figure 6. Overview of view synthesis using a re-rendering network. The 3D reconstruction is equipped with trainable latent codes which are rasterized into a graphics buffer. The graphics buffers are translated into the target view using the re-rendering network.

3.3. Volumetric Representations

In this section, we review a line of work in learning-based novel view synthesis where the scene is stored in an intermediate volumetric representation. Next, we will review two main categories of such representations, multi-plane images and voxel grids.

3.3.1. Multi-Plane Images

A multi-plane image is a stack of RGBα images that are placed along a view frustrum at a fixed range of depths, where α is the opacity of the pixel. The way these images are positioned in space is demonstrated in Figure 7.

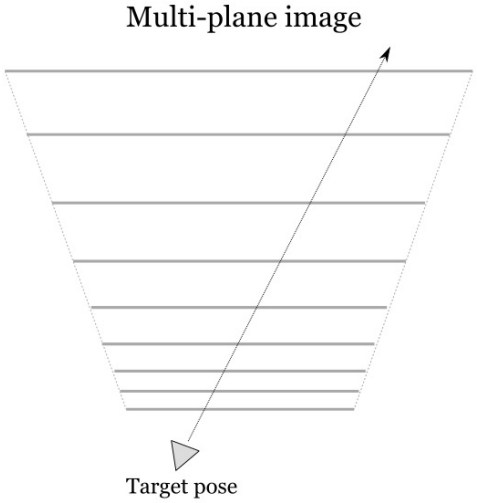


Figure 7. 2D-projection of the multi-plane image representation.

The multi-plane image representation (MPI) was popularized by Zhou et. al. [36] as a way to synthesize views from forward facing scenes. Given a camera ray that intersects each image at depth $d = 1, \cdots, D$, the projected color $\mathbf{c}$ is formed by compositing each intersected color $\mathbf{c}_d$ in front-to-back fashion:

$$\mathbf{c} = \sum_{d=1}^{D} \alpha_d \mathbf{c}_d \prod_{i=d+1}^{D} (1 - \alpha_i) \tag{8}$$

Zhou et. al. demonstrated that using a homography to warp each plane in the MPI to a target viewpoint and compositing the warped planes with Equation 8 is an effective way to synthesize views in a differentiable manner. Their method involves training a CNN to predict an MPI from stereo images. The input to their CNN is a stereo image pair, in which the camera pose of the second image is implicitly encoded as input by computing a plane sweep volume that reprojects it to the first image. The network is trained in end-to-end fashion by predicting the MPI and projecting it to a target image using a homography and alpha compositing as in Equation 8. This method is summarized in Figure 8.

Subsequent work [37] [38] addresses the fact that MPIs alone cannot model view-dependent effects in a reliable way. Specifically, Mildenhall et. al. [37] proposed to blend between multiple MPIs and [37] Wizadwongsa et. al. [38] proposed to store linear combinations of basis functions in the MPI instead of colors [38].

In addition to multi-plane images, other researchers have considered alternative image formats such as spherical [39] and cylindrical [40] image representations to synthesize views from outward facing 360° scenes. This approach is in essence similar except for a different warping scheme and finds use in a stereo setup of 360° cameras. An example of a multi-sphere image is visualized in Figure 9.

The key advantage of multi-plane images is that they can be readily stored in image formats and integrated to graphics engines for real-time rendering without the need to evaluate a neural network. However, these representation are relatively limited in terms of translational movement in the scene.

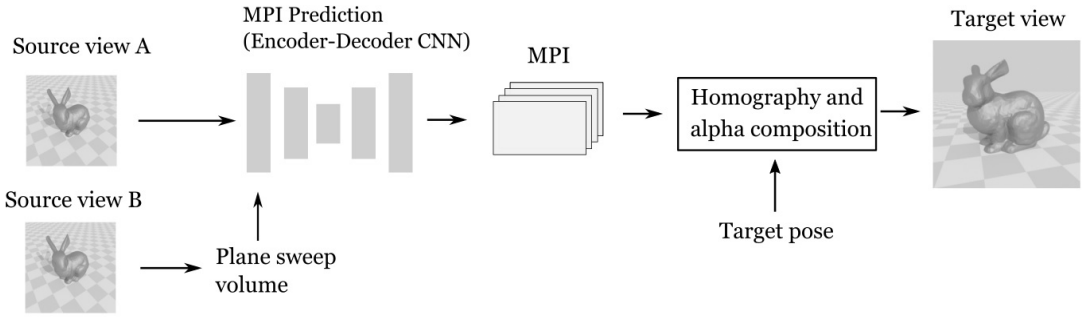


Figure 8. MPI prediction and view synthesis with stereo images: A stereo image pair is sent to an encoder-decoder CNN that predicts color and alpha planes. The camera pose of the second image is implicitly encoded as input by computing a plane sweep volume that reprojects it to the first image. The target view is synthesized by projecting the MPI to the target pose via a homography, then compositing the projected colors with the alpha values.

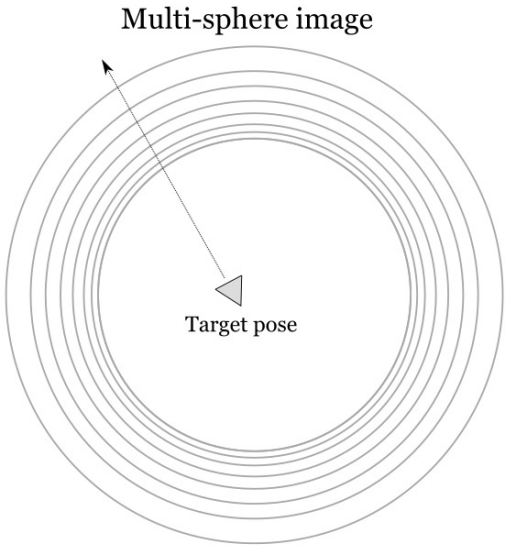


Figure 9. 2D-projection of the multi-sphere image representation.

3.3.2. Voxel Grids

A voxel grid stores information about the scene in a cartesian grid-like structure. Unlike multi-plane images, voxel grids are not limited to forward facing scenes and can potentially describe any type of scene within a bounded volume.

Sitzmann et. al. [41] and Nguyen et. al. [42] proposed to use convolutional neural networks to encode images into a voxel grid of latent codes and project the grid representation back to a novel viewpoint. However, the problem with this approach is that the grid is very limited in terms of resolution to fit all of the latent codes in memory. Later, Nguyen et. al. [43] demonstrated a method to compose scenes by encoding each object to a separate voxel grid.

Instead of using a convolutional network to project latent codes, Lombardi et. al. [44] introduced a method where a convolutional network predicts a RGBα grid, which is then rendered to a novel view via volume rendering. In addition, they address the memory limitation of the voxel grid by accessing the volume indirectly using a warp field. More recently, others have experimented with sparse voxel octrees [45], illustrated in Figure 10, to store and render voxels in a more efficient manner [46] [47]. While sparse voxel octrees are more challenging to work with than cartesian grids, they take significantly less memory to store and are faster to render.

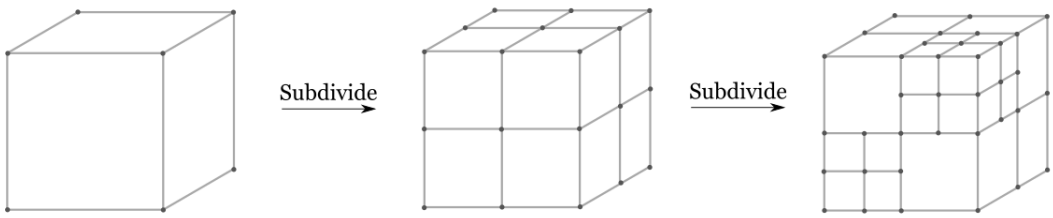


Figure 10. An octree is constructed by recursively subdividing a volume into 8 octants while pruning octants that contain empty space.

3.4. Implicit Neural Representations

Implicit neural representations store information about the scene in the weights of a neural network. In this section, we review the concept of coordinate-based neural representations and how they are utilized in novel view synthesis in the form of radiance fields, which have become a promising research direction in neural scene representations. Finally, we look at other neural scene representations used in NVS.

3.4.1. Coordinate-Based Neural Representations

Coordinate-based neural representations have recently seen attention as an alternative to discrete 3D-representations such as point clouds to describe 3D shapes. Coordinate-based neural representations are networks that are conditioned on spatial coordinates such as position and direction which output a continuous quantity that implicitly describes the geometry of the scene. Early work in this area considered occupancy

fields [48] as well as signed distance fields [49] where the geometry is defined as the level-set of these quantities. This work, however, was restricted to explicit 3D information, e.g, point clouds or meshes, as the supervisory signal. Later on, more advanced differentiable rendering networks [50] [51] enabled to use images as the supervisory signal. Finally, the introduction of radiance fields paired with volume rendering was found to be a promising direction for differentiable rendering [18]. It was speculated that volume rendering widens the basin of convergence, enabling the discovery of good solutions [44].

A common choice of network architecture for these representations are multi-layer perceptrons with ReLU activation. While ReLU is computationally convenient, it was found that using ReLU for this purpose causes the network to struggle at modeling high frequency content [52] [17]. To counter this issue, Tancik et. al. [52] showed that mapping the input coordinates to a frequency domain $\gamma(x)$ as a pre-processing step lets the network converge much more rapidly:

$$\gamma(x) = (\cos(2^0\pi x), \sin(2^0\pi x), \cdots, \cos(2^{L-1}\pi x), \sin(2^{L-1}\pi x)) \tag{9}$$

This mapping, known as a positional encoding, was found to increase the performance of MLPs at expressing low-dimensional functions. The parameter L controls the amount of under- and overfitting, ideally chosen to achieve a balance between the two extremes. Note that in Equation 9, x should be normalized to $[-1, 1]$. Alternatively, Sitzmann et. al. showed that using a sine activation function in place of ReLU achieves similar results [17]. A summary of coordinate-based neural representations using the above mapping is shown in Figure 11.

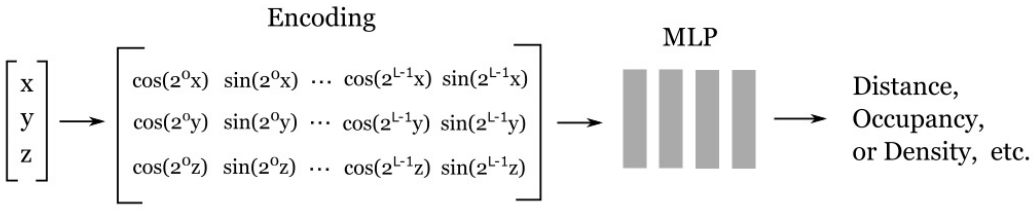


Figure 11. Coordinate-based neural representations can model the scene as an implicit function and synthesize views using a differentiable rendering scheme.

The disadvantage of using MLPs to model scenes is that training and evaluating them is rather time consuming, as they need to be trained for each scene separately. Recent research tries to address the slow training by using learned initializations [53] and the slow evaluation by caching the scene into volumetric representations after training [47] [54]. Since MLPs are continuous, they are not limited to a specific resolution and instead their memory footprint scales in terms of signal complexity.

3.4.2. Neural Radiance Fields

Neural radiance fields are a class of coordinate-based neural representations that have recently seen attention as a simple and effective method for differentiable rendering.

Encoding a radiance field within the weights of a neural network was popularized by Mildenhall et. al. [18]. A neural radiance field encodes the scene as a function of position $\mathbf{x}$ and view direction $\mathbf{d}$, returning the predicted radiance and density $(\mathbf{C}, \sigma)$ at that point. Rendering an image from a radiance field involves volume rendering. Given a camera ray $\mathbf{r}(t)$, the synthesized color is evaluated by approximating a volumetric integral previously known from classical volume rendering [55]:

$$\mathbf{c} = \int_{t_{\text{near}}}^{t_{\text{far}}} T(t)\sigma(\mathbf{r}(t))\mathbf{C}(\mathbf{r}(t))\, dt \tag{10}$$

$$T(t) = \exp(-\int_{t_{\text{near}}}^{t} \sigma(\mathbf{r}(t))\, dt) \tag{11}$$

Both of these integrals can be approximated with a finite sampling scheme and hence the method is end-to-end differentiable. The process of synthesizing a color with a neural radiance field is demonstrated in Figure 12.

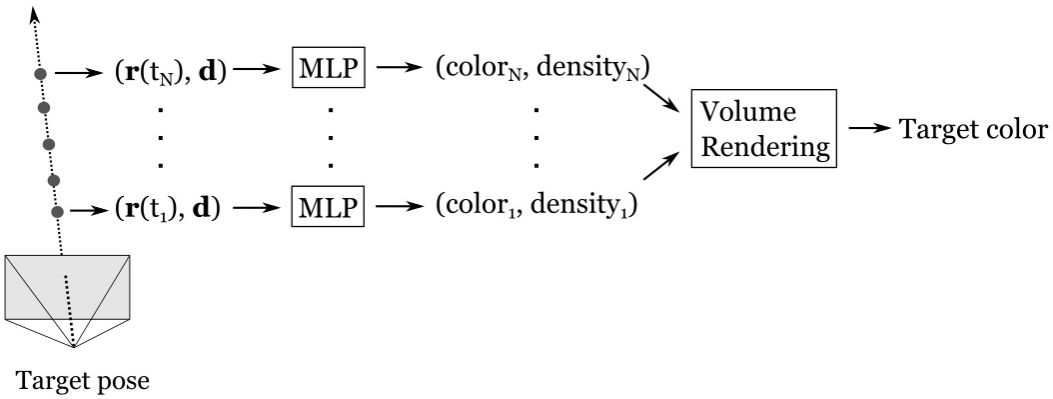


Figure 12. Volume rendering with a neural radiance field parametrized by a MLP. At each sample along the camera ray, the position and direction coordinates are encoded as in Equation 9 and sent to a MLP to predict color and density. The colors and densities at each sample are used to approximate the volume integral in Equation 10 which is the predicted target color.

Later on, a vast amount of research has been dedicated on improving and extending neural radiance fields to various directions.

Acceleration: A number of works address the slow rendering time, since evaluating a multi-layer perceptron at each point in the ray is rather time consuming. Liu et. al. [46] proposed to decompose the radiance field into a sparse voxel octree where each vertex stores a local radiance field. Others have noted that a radiance field can be factorized into position- and view dependent components. This greatly reduces the memory requirements when stored in a volumetric representation, enabling real-time rendering [54] [47].

Robustness: Martin et. al [56] and Zhang et. al. [57] improve the original model at synthesizing views in challenging scenarios such as scenes with photometric inconsistencies, occluders [56] and scenes with unbounded background content [57].

Strong prior: Some authors note that the original model fails to synthesize views from scenes with only a few images, and address this issue by imposing a strong prior

on the scene by encoding each pixel with additional features using CNN-architectures. [58]

Relighting and scene composition: Srinivasan et. al. [59] experiment with explicit light scattering models, in which the volume integral is more akin to the rendering equation [60] , enabling ways to relight the scene. Guo et. al. [61] address scene composition by fitting a radiance field separately to each object and demonstrating how to compose radiance fields.

Videos and dynamic scenes: A number of authors note that radiance fields can be conditioned on time, enabling view synthesis on dynamic scenes by training the network on video content [62] [63].

Camera estimation: Finally, Wang et. al. [64] experiment with a method to jointly determine the radiance field and camera poses from a set of images by converting the intrinsic and extrinsic parameters of the virtual camera into trainable parameters.

3.4.3. Other Neural Representations

Finally, some authors have considered implicit neural representations that encode a fully abstract description of the scene [65] [25]. The Generative Query Network introduced by Eslami et. al. [65] is a encoder-decoder type of architecture that takes in a set of posed images as input and produces a scene representation by averaging each encoded representation. The decoder is conditioned on a camera pose to synthesize an image from a novel viewpoint and interestingly, they demonstrate that scenes can be composited and edited by performing algebraic operations on different scene representations. Nguyen et. al. [25] improve on this method by using a Transformer architecture to predict views in a sequential manner.

4. DATA COLLECTION

In this chapter, we describe the process of collecting and preparing multi-view stereo datasets for novel view synthesis and present the datasets that were collected for the purposes of this thesis.

4.1. Data Collection Process

When collecting multi-view stereo datasets, the goal is to capture a consistent set of images from the scene and to obtain an accurate estimate of the intrinsic and extrinsic parameters of the camera associated with each image.

4.1.1. Image Acquisition

In order to capture images from the scene in a consistent way, it is worth looking at some of the properties of the imaging device and how they relate to multi-view consistency:

ISO: ISO is a measure of how much the electronic signal of the image sensor is amplified. A high ISO value can produce noisy images and therefore, ISO should be kept low to reduce noise in the image. A well lit environment is essential for capturing bright images with low ISO.

Shutter speed: Shutter speed is the length of time the camera sensor is exposed to light. Using a long shutter speed can lead to blurry images, which is not desirable for consistent results.

Autoexposure: Autoexposure is a setting that automatically adjusts ISO and shutter speed to keep the brightness of the image constant. For consistent results, it is essential to disable autoexposure for the duration of the capture.

Focus distance and autofocus: Focus distance is a property of the lens that affects the distance in which objects in the scene are in focus, i.e. not blurry. Autofocus is a setting that automatically adjusts the focus distance. This affects the focal length of the lens and breaks the assumption that camera parameters stay fixed during the capture. Using a fixed focus distance should be preferred if possible.

Image format: Finally, exporting the images in a compressed format, such as JPG, should be avoided, and instead prefer uncompressed formats such as PNG and RAW.

As for this thesis, we use a Samsung A50 mobile phone as the choice of camera and the OpenCamera Android application to capture images, shown in Figure 13. For each scene, we follow the above guidelines, keeping autoexposure and focus distance fixed. However, the images are stored in JPG format because of limitations imposed by the manufacturer. The details of the camera are summarized in Table 1.

Megapixels	Image width	Image height	Aperture	Focal length (mm)
25	5685	4249	f/1.7	26

Table 1. Specifications for the camera used in this thesis (Samsung A50).

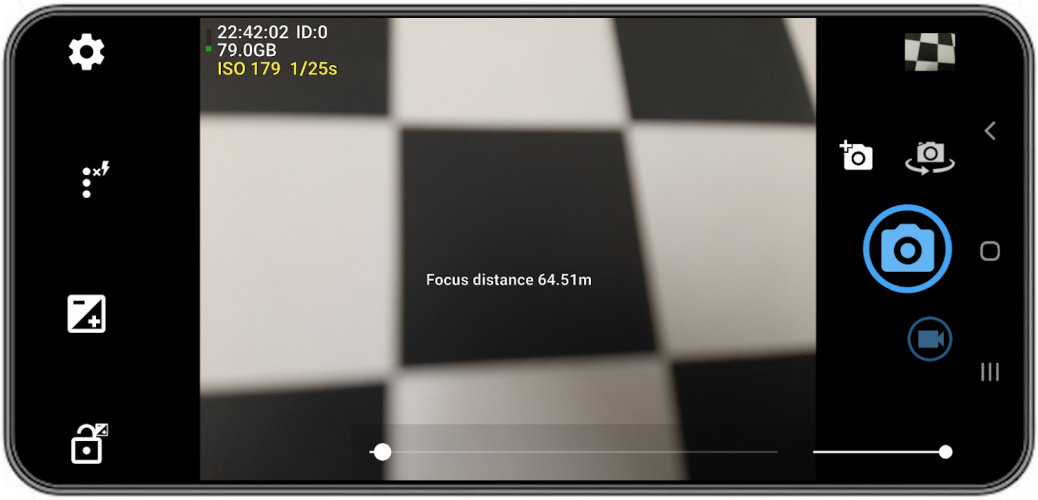


Figure 13. The OpenCamera Android application. Top left corner: ISO 179 and shutter speed 1/25 [s]. Bottom left corner: Option to keep ISO and shutter speed fixed. Bottom center: Slider to change the focus distance.

4.1.2. Calibration

Most novel view synthesis algorithms work under the assumption that the extrinsic and intrinsic parameters of the camera corresponding to each image are known. Next, we review the meaning of intrinsic parameters and describe the process of finding the intrinsic parameters of the camera without prior information.

The intrinsic parameters describe how a point $(x, y, z)^T$ in the camera coordinate system is projected to the image plane coordinates (u, v). In an ideal pinhole model, this projection is parametrized by focal lengths f_x, f_y and principal point offset c_x, c_y:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} f_x x' + c_x \\ f_y y' + c_y \end{pmatrix} = \begin{pmatrix} f_x \frac{x}{z} + c_x \\ f_y \frac{y}{z} + c_y \end{pmatrix} \tag{12}$$

However, the ideal pinhole model is a simplification as lenses have a varying amount of distortion. To better describe the way real-world lenses project, we add additional degrees of freedom to the camera model. Here, we adopt a distortion model with two degrees of freedom for radial distortion k and tangential distortion p:

$$x'_{\text{distorted}} = x(1 + k_1 r^2 + k_2 r^4) + (2p_1 xy + p_2^2(r^2 + 2x^2)) \tag{13}$$

$$y'_{\text{distorted}} = y(1 + k_1 r^2 + k_2 r^4) + (2p_2 xy + p_1^2(r^2 + 2y^2)) \tag{14}$$

Where $r^2 = (x')^2 + (y')^2$. Substituting the above to Equation 12, we have

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} f_x x'_{\text{distorted}} + c_x \\ f_y y'_{\text{distorted}} + c_y \end{pmatrix} \tag{15}$$

Obtaining these parameters can be done using a point correspondence algorithm, where the idea is to take several pictures of an object with known 3D correspondences,

for example a chessboard, and minimize a re-projection error with respect to the parameters. For the camera used in this thesis, we obtained the parameters by taking 50 images of a chessboard and using an open source implementation of the above mentioned point correspondence algorithm [66]. The related script for obtaining the intrinsic parameters is presented in Appendix A. In Figure 14 we see a few examples of the images used for point correspondence, and in Table 2 the obtained parameters.

f_x	f_y	c_x	c_y	k_1	k_2	p_1	p_2	RMS [px]
4426	4435	2881	2202	0.091	-0.209	0.006	0.000	1.158

Table 2. Estimated camera coefficients and reprojection error.

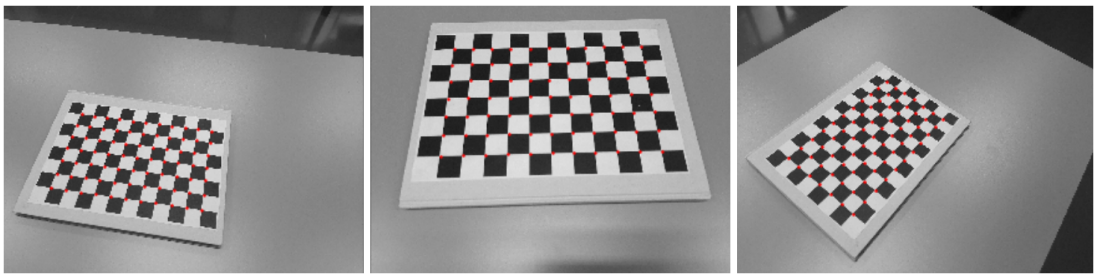


Figure 14. A few examples of the chessboard calibration images. The detected corner points are shown as red dots.

4.1.3. Structure from Motion

Structure from motion is a widely used algorithm to find the extrinsic parameters of the camera, i.e. camera poses, corresponding to each image. In this thesis, we use COLMAP [5] which is an open-source implementation of SfM. COLMAP can also be used to refine the estimated intrinsics, estimate intrinsics without prior information, undistort images and generate a 3D reconstruction of the scene.

For each captured scene, we provide COLMAP with the images and estimated intrinsics and run the algorithm with default settings. After obtaining the camera poses, we let COLMAP undistort the images with the refined intrinsics, which converts the images into an idealized pinhole model by solving Equation 13 and 14. This is important because most novel view synthesis algorithms assume that the images originate from an ideal pinhole camera model. In Figure 17, we see an example of camera motion estimated by COLMAP for a forward-facing scene and the reconstructed point cloud and in Figure 18, we see an example of how COLMAP can create a 3D reconstruction of the scene.

4.2. Collected Datasets

The goal of the data collection was to capture a handful of different types of scenes, including forward facing scenes (Figure 17, Left) and inward facing scenes (Figure 17, Right). All of the scenes have different background and some of them have more

challenging geometries and materials, such as shiny objects. A total of 6 scenes were collected with 30-90 images each, summarized in Table 3. The file structure of the dataset is seen in Figure 15.

Name	# Train	# Val.	Description
Classroom 1	42	6	A forward-facing scene of a classroom.
Classroom 2	30	6	A forward-facing scene of a classroom.
Cafeteria	42	6	A forward-facing scene of a cafeteria.
Bike	66	6	An inward-facing scene of a bike.
Printer	48	6	An inward-facing scene of a printer.
Vase	90	6	An inward facing-scene of a glass vase.

Table 3. A summary of the collected scenes, including the number of images used for training and evaluation.

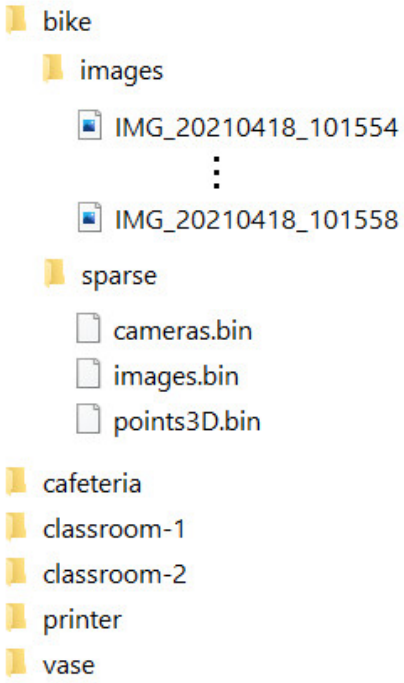


Figure 15. File structure of the collected dataset. "cameras.bin" contains the intrinsic parameters of the camera, "images.bin" contains a list of filenames corresponding to each image and "points3D.bin" contains the camera pose and a list of 3D points associated with each image.

Figure 16. Example image from each scene. From left to right: Classroom 1, Classroom 2, Cafeteria, Bike, Printer, Vase.

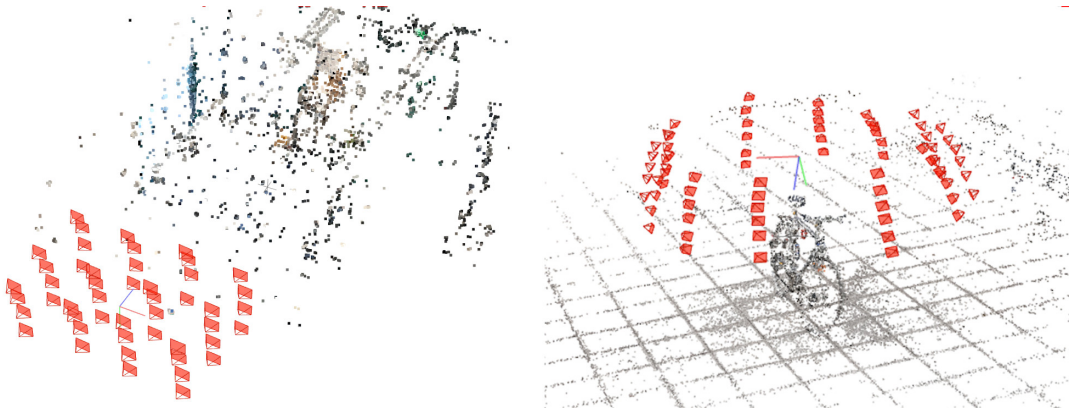


Figure 17. Screenshots from COLMAP showing the pattern of camera motion and point cloud for a forward-facing scene (Left) and an inward-facing scene (Right).

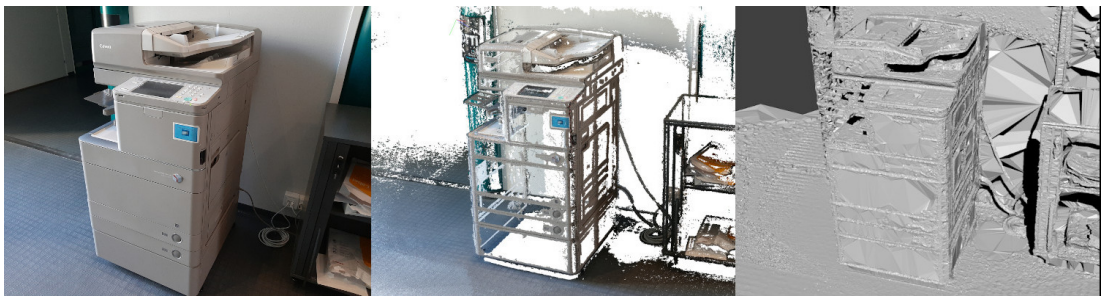


Figure 18. Left: Example image from the "printer" scene. Middle: Dense point cloud reconstruction by COLMAP, Right: A 3d mesh obtained from the point cloud via Delaunay triangulation

5. EXPERIMENTS

In this chapter, we describe the setting in which we evaluate and experiment with state of the art methods in learning-based novel view synthesis. The experiments are carried out with carefully selected and recently published open-source methods, which are NeRF [18], NeRF−− [64], NeX [38] and IBR-Net [26]. The reasoning behind these choices is a combination of availability, time limitations, date of release and functionality. Most importantly, each one have unique characteristics that relate to the discussed methodologies in learning-based novel view synthesis, namely, image-based rendering, volumetric representations and implicit neural representations.

5.1. NeRF and NeRF--

5.1.1. Overview

The first view synthesis method that is adopted for experimentation is NeRF[1][18], which falls into the class of coordinate-based neural representations and specifically neural radiance fields discussed in Section 3.4.2. Synthesizing a novel view with NeRF involves training a multi-layer perceptron that returns radiance $\mathbf{C}$ and density σ for a given position and view direction in the scene. NeRF synthesizes a target color $\mathbf{c}$ by querying the MLP along a camera ray $\mathbf{r}(t)$ and evaluating the volume integral in Equation 10 using the following approximation:

$$\mathbf{c} \approx \sum_{i=1}^{N} T(t_i)\alpha_i \mathbf{C}(\mathbf{r}(t_i)) \tag{16}$$

$$\alpha_i = 1 - e^{-\sigma(\mathbf{r}(t_i))\delta} \tag{17}$$

$$T(t_i) = \prod_{j=1}^{i} 1 - \alpha_i \tag{18}$$

The samples t_i lie along the camera ray and are drawn from a stratified uniform distribution to increase coverage over multiple training iterations. Furthermore, NeRF chooses to optimize two different networks, a coarse and a fine network, in which the coarse network samples the radiance field in the above described manner and the fine network samples t_i from a distribution that is proportional to previously obtained weights, i.e. $p_i = \frac{T_i\sigma_i}{\sum_i T_i\sigma_i}$. This lets the fine network allocate the majority of samples in locations that are most likely to contribute to the integral. The parameters of the model are optimized with stochastic gradient descent, specifically ADAM, against the ground truth pixel value $\mathbf{c}_{gt}$.

$$\mathcal{L} = ||\mathbf{c} - \mathbf{c}_{gt}||_2^2 \tag{19}$$

[1] https://github.com/yenchenlin/nerf-pytorch

The second method to be adopted for experimentation is NeRF−−[2][64], which builds upon the original NeRF by parametrizing the camera model in addition to the network weights to jointly optimize the radiance field and camera poses. Note that in this thesis, we use COLMAP for acquiring the camera poses and this approach is purely used for experimentation. Specifically, NeRF−− parametrizes the origin, rotation and focal lengths of the camera associated with each image. The view direction associated with a color at coordinates u, v is then obtained using the standard pinhole camera model:

$$\mathbf{d}_{u,v} = \hat{\mathbf{R}}_i(\frac{u - \frac{W}{2}}{\hat{f}_x}, \frac{v - \frac{H}{2}}{\hat{f}_y}, -1)^T \tag{20}$$

Here (W, H) is the resolution of the image and $(\hat{f}_y, \hat{f}_y)$ are the parametrized focal lengths. The rotation matrix $\hat{\mathbf{R}}_i$ needs to be parametrized as a function of axis and angle, which is conveniently done using Rodrigues' rotation formula:

$$\hat{\mathbf{R}} = \mathbf{I} + \frac{\sin(\alpha)}{\alpha} \begin{pmatrix} 0 & -\psi_2 & \psi_1 \\ \psi_2 & 0 & -\psi_0 \\ -\psi_1 & \psi_0 & 0 \end{pmatrix} + \frac{1 - \cos(\alpha)}{\alpha^2} \begin{pmatrix} 0 & -\psi_2 & \psi_1 \\ \psi_2 & 0 & -\psi_0 \\ -\psi_1 & \psi_0 & 0 \end{pmatrix}^2 \tag{21}$$

Where $(\alpha, \psi_0, \psi_1, \psi_2)$ are the trainable parameters.

5.1.2. Network Architecture

The network architecture of NeRF is a multi-layer perceptron that takes as input the position and direction coordinates along a camera ray. The coordinates are encoded as described in Equation 9, with the choice of $L = 10$ for position coordinates and $L = 4$ for view direction coordinates in this thesis. The model predicts a color and density value for the given position and direction encodings, which are then integrated along the ray to predict the final color as in Equation 16. The MLP architecture is shown in Figure 19. A minimal example of this model written in PyTorch is presented line by line in Appendix B. In NeRF−−, the network architecture is the same, except the position and direction encodings are dependent on the parameters of the camera model.

5.1.3. Training Details

Iterations, samples and learning rate: The NeRF model was trained on the collected datasets with 4096 rays per iteration, for a varying number of iterations per scene, shown in Table 4. The coarse network was trained with 64 samples and fine network with 128, both with initial learning rate of $0{,}5 * 10^{-4}$ decayed over the course of training.

Inverse depth sampling experiment: To explore the effect of different sampling strategies, the Bike scene was trained additionally without inverse depth sampling, in

[2] https://github.com/ActiveVisionLab/nerfmm

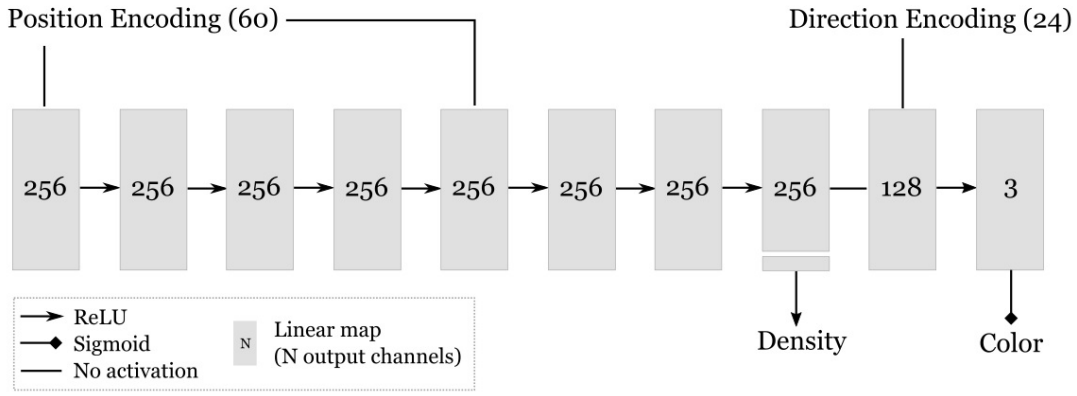


Figure 19. The MLP architecture of NeRF used in this thesis. The input to this network are the position and direction coordinates of the camera ray at a specific sample, where each coordinate is normalized between $[-1, 1]$ and encoded as in Equation 9. In this thesis, we use $L = 10$ for position coordinates and $L = 4$ for view direction coordinates, for a total dimension of 60 for positional encoding and 24 for directional encoding.

which samples are drawn linearly in disparity rather than depth. Note that in all other scenarios inverse depth sampling is used.

Calibration experiment: To explore the practical advantage of manual camera calibration, the Classroom 2 scene was trained additionally without providing the estimated camera intrinsics (Table 2) to COLMAP, resulting in slightly different camera poses.

NeRF$--$: The NeRF$--$ model was trained in 3 different settings for the Classroom 2 scene. In the first scenario, the poses are set to identity. In the second scenario, the poses are fixed to poses estimated by COLMAP. In the third scenario, the poses estimated by COLMAP are refined by NeRF$--$. The model was trained with 1024 rays per image for 4000 epochs.

Scene	Iterations	Image scale
Classroom 1	200000	0.250
Classroom 2	80000	0.250
Cafeteria	80000	0.250
Printer	80000	0.250
Bike	200000	0.250
Vase	200000	0.167

Table 4. Total iterations and image downscale factor for each scene trained with NeRF.

The training loss over time for the Classroom scene is plotted in Figure 20. For other scenes the loss converged in a similar way, omitted for brevity. The training process took roughly 18 hours for Classroom 1, Bike and Vase, and roughly 8 hours for Classroom 2, Cafeteria and Printer, using a single Tesla P100 graphics card.

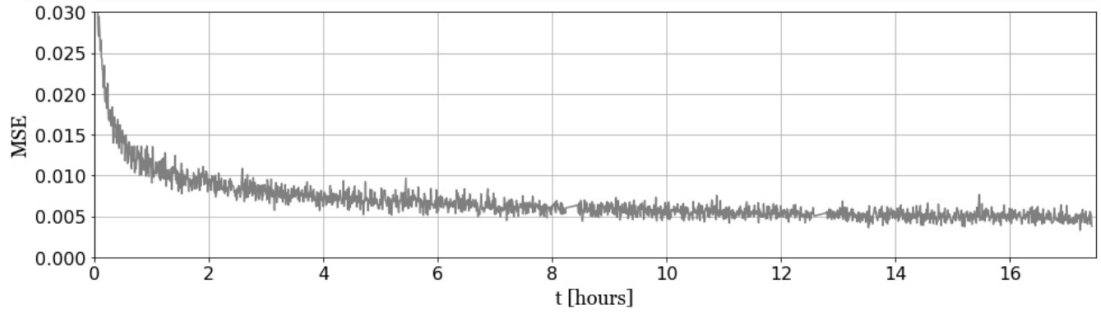


Figure 20. NeRF: Measured MSE over time for the Classroom scene.

5.2. NeX

5.2.1. Overview

The third view synthesis method that is adopted for experimentation is NeX[3][38], which is characterized by it's volumetric scene representation and specifically the multi-plane image discussed in Section 3.3.1. Instead of directly storing color information in the MPI, NeX chooses to store position-dependent basis coefficients $\mathbf{k}(x, y, d)$ and view direction dependent basis functions $\mathbf{b}(\boldsymbol{\omega})$ separately as a way to store view-dependent effects, where (x, y) are the image coordinates, d is the plane depth and $\boldsymbol{\omega}$ is the view direction. The view-dependent color $\mathbf{c}$ is then regressed as a linear combination of these values:

$$\mathbf{c} = \mathbf{k}_0(x, y, d) + \sum_{i=1}^{N} \mathbf{k}_i(x, y, d)\mathbf{b}_i(\boldsymbol{\omega}) \tag{22}$$

Training NeX for view synthesis involves using a homography to warp each plane in the MPI representation to a target viewpoint. After applying the homography, the image coordinates (x, y) and plane depth d of the warped planes are sampled and sent to a multi-layer perceptron that returns opacity and basis coefficients. Similarly, the view direction is encoded and sent to another MLP that predicts basis functions. This process is summarized in Figure 21. Finally, the target view is synthesized by regressing the color (Equation 22) and using alpha composition as in Equation 8. In this thesis, we use a total of 16 planes for the MPI and a total of 8 basis functions.

The parameters of the model are optimized with ADAM using L2-loss between the ground-truth image I_{gt} and synthesized image I, including a regularization term with the respective gradients:

$$\mathcal{L} = ||I_{\text{gt}} - I||_2^2 + \lambda||\nabla I_{\text{gt}} - \nabla I||_2^2 \tag{23}$$

After training, the obtained basis coefficients and basis functions can be stored explicitly in images and integrated to graphics engines, such as WebGL, for fast rendering. This way, the views can be synthesized by only using a homography, color regression and alpha composition.

[3] https://github.com/nex-mpi/nex-code

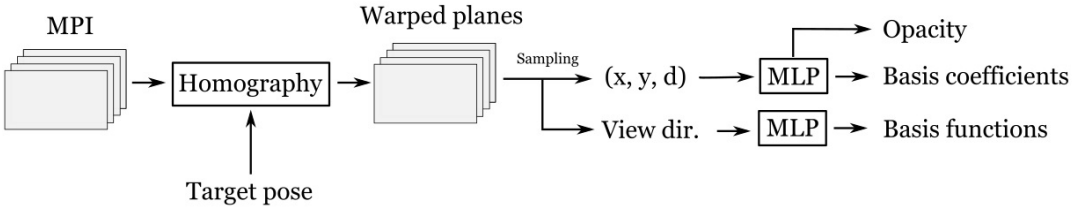


Figure 21. Training NeX for view synthesis: The MPI coordinate system is projected to a target pose via a homography warping. The warped planes are sampled at image coordinates (x, y) and plane depth d, which are encoded as in Equation 9 and sent to a MLP that predicts basis coefficients and opacity. Similarly, the view direction is encoded and sent to another MLP that predicts basis functions. The colors at each plane depth d are regressed with Equation 22 and composited to the target color with alpha composition (Equation 8).

5.2.2. Network Architecture

The network architecture of NeX is a factorized multi-layer perceptron that takes as input the image coordinates (x, y), plane depth d and view direction $\boldsymbol{\omega}$. The coordinates are encoded as described in Equation 9, with the choice of $L = 10$ for image coordinates, $L = 8$ for depth and $L = 3$ for view direction coordinates in this thesis. The model predicts the basis functions and basis coefficients separately which are then used to predict the color as in Equation 22. The MLP architecture is shown in Figure 22.

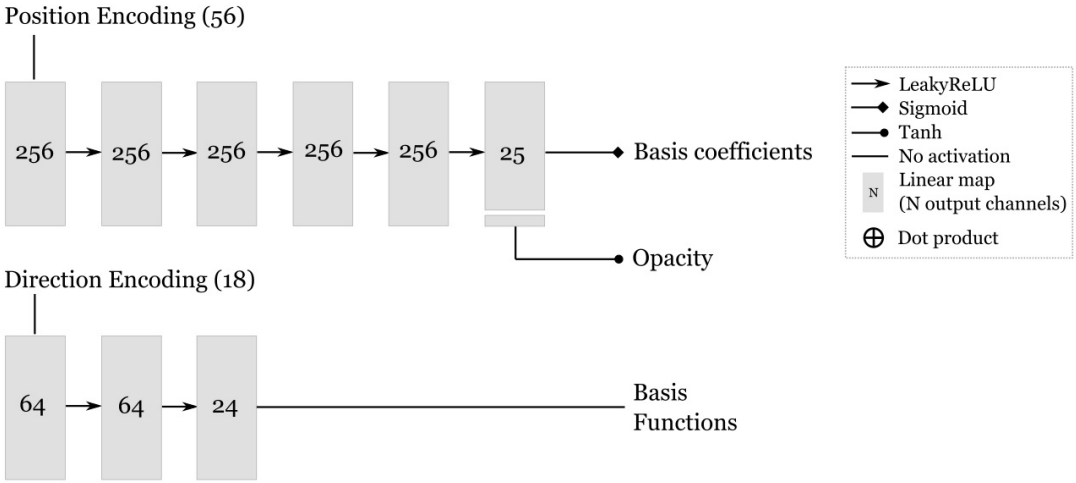


Figure 22. The MLP architecture of NeX used in this thesis. The input to this network are the image coordinates and plane depth, denoted together as position, and the view direction coordinates. Each coordinate is normalized between $[-1, 1]$ and encoded as in Equation 9. In this thesis, we use $L = 10$ for image coordinates, $L = 8$ for depth and $L = 3$ for view direction, for a total dimension of 56 for positional encoding and 18 for directional encoding.

5.2.3. Training Details

The NeX model was trained on each of the collected scenes using 8000 rays per image for 1000 iterations, with a learning rate of 10^{-3} decayed over the course of training. The training loss over time for the Classroom scene is plotted in Figure 23. For other scenes the loss converged in a similar way, omitted for brevity. The training process took roughly 9 hours for each scene using a single Tesla P100 graphics card.

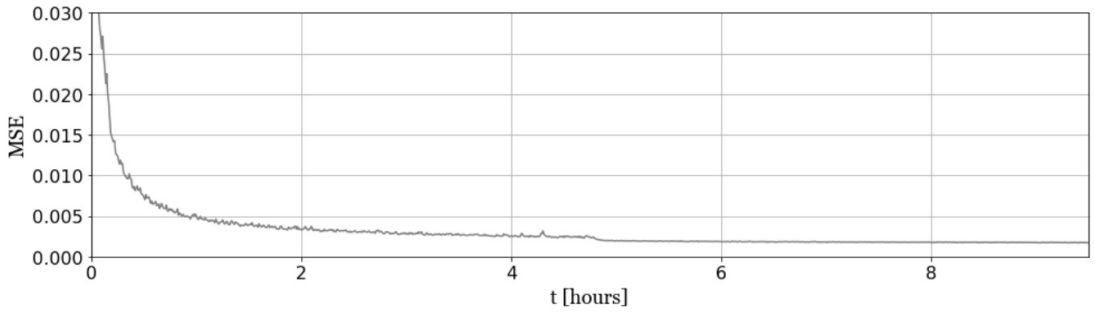


Figure 23. NeX: Measured MSE over time for the Classroom scene.

5.3. IBR-Net

5.3.1. Overview

The final method that is adopted for experimentation is IBR-Net[4][26]. IBR-Net can be categorized as neural image-based rendering discussed in Section 3.2.1 in the sense that it directly aggregates information from nearby images to synthesize views. Similar to NeRF, synthesizing a novel view with IBR-Net involves volume rendering along a camera ray. At each sample along the ray, IBR-Net aggregates information from each source view which are sent to a network that predicts color and density features. The density features are sent to a Transformer that predicts density at each sample. This process is illustrated in Figure 24. The density and color are then integrated along the camera ray as in Equation 16 to form the synthesized color. Furthermore, IBR-Net optimizes two different networks, a coarse and a fine network, in the same way as in NeRF discussed in Section 5.1.1. The rendering loss function of IBR-Net is also the same as in NeRF.

5.3.2. Network Architecture

Compared to NeRF and NeX, the network architecture of IBR-Net (MLP*) is more involved, containing several sub-networks. Starting from the source images, each image is sent to a ResNet-34 [20] to extract image features. The aggregated image features are then sent to a network illustrated in Figure 25 which outputs color and density features F_i. Finally, the density features are sent to a Transformer which outputs the final density.

[4] https://github.com/googleinterns/IBRNet

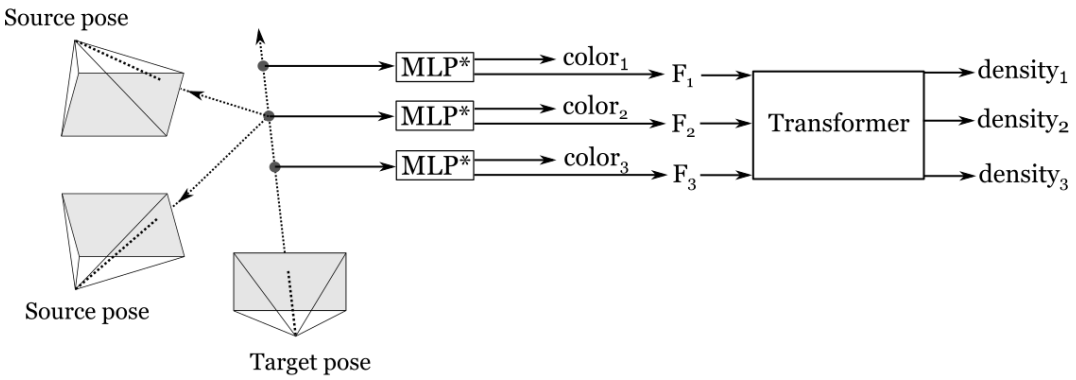


Figure 24. Overview of IBR-Net. At each sample along the ray, IBR-Net aggregates information from each source view which are sent to the denoted MLP* network, explained in more detail in Section 5.3.2, that predicts color and density features. The density features are sent to a Transformer which outputs the final density at each sample.

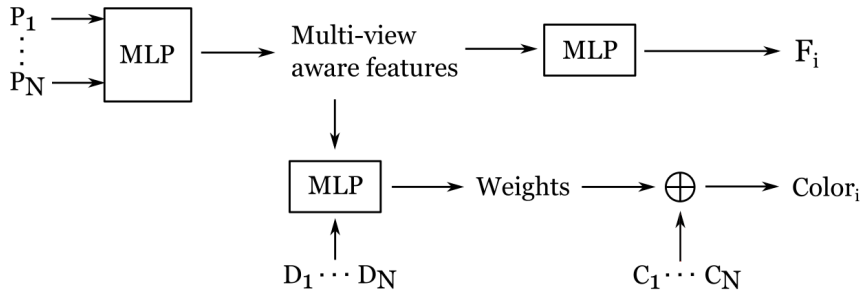


Figure 25. The MLP* -network. The image features P are sent to a MLP that outputs multi-view aware features. These features are then sent to another MLP along with relative view directions D to produce blending weights. Finally, the blending weights are combined with the corresponding source colors C to form the predicted color.

5.3.3. Training Details

In this thesis, we used a pre-trained IBR-Net that was trained on a variety of multi-view stereo datasets. The pre-trained network was reportedly trained on eight V100 GPUs for about a day, which should be considerably more FLOPS than we used for NeRF and NeX. As for inference, the number of aggregated source views was fixed at 10 and the amount of coarse and fine samples were set to 64. Additionally, the Cafeteria scene was evaluated with varying amount of aggregated source views (2, 4, 6, 8, 10) to explore how this affects the quality of synthesized views.

6. EVALUATION

In this chapter, we present the results of the conducted experiments and compare the results quantitatively and qualitatively. First however, we review the adopted metrics used for evaluating the selected methods.

6.1. Metrics

Measuring the success of a view synthesis algorithm to produce plausible and realistic views requires a metric of perceptual similarity between a synthesized and a ground truth image. The metrics that are adopted in this thesis are are peak-to-signal noise ratio (PSNR), structural similarity index (SSIM) [67] and learned perceptual image patch similarity (LPIPS) [68].

6.1.1. PNSR

The first one of these, PSNR, is a commonly used metric to measure the reconstruction quality of images and is directly related to the mean squared error (MSE). Denoting image resolution as (W, H) and assuming that each color lies in $[0, 1]^3$, the MSE and PSNR between a ground truth RGB-image I^{gt} and a synthesized RGB-image I is defined as:

$$\text{MSE} = \frac{1}{3WH} \sum_{j=1}^{H} \sum_{i=1}^{W} ||I_{i,j} - I^{\text{gt}}_{i,j}||_2^2 \tag{24}$$

$$\text{PSNR} = 20 \log_{10}(\max(I)) - 10 \log_{10}(\text{MSE}) = -10 \log_{10}(\text{MSE}) \tag{25}$$

6.1.2. SSIM

The second metric, SSIM, improves on measuring perceptual similarity by noting that the human perceptual system is highly sensitive to variation in structural information. SSIM models the structure, contrast and luminance between images by using statistical features, specifically, the mean μ_I, variance σ_I^2 and covariance $\sigma_{I^{\text{gt}},I}$ of both images:

$$\text{SSIM} = \frac{(2\mu_{I^{\text{gt}}}\mu_I + C_1)(2\sigma_{I^{\text{gt}},I} + C_2)}{(\mu_{I^{\text{gt}}}^2 + \mu_I^2 + C_1)(\sigma_{I^{\text{gt}}}^2 + \sigma_I^2 + C_2)} \tag{26}$$

To clarify, C_1 and C_2 are small constants to prevent divergence. The above expression is computed locally using a gaussian window of size 11x11 and averaged for each region to get the final result. In this thesis, we use an open source implementation of SSIM[5] [69] where $C_1 = 0{,}01$ and $C_2 = 0{,}03$.

[5] https://www.tensorflow.org/api_docs/python/tf/image/ssim

6.1.3. LPIPS

The final metric, LPIPS, is based on comparing distances between feature representations obtained from the layers of a deep neural network. To give some background, it was discovered that DNNs trained on image prediction and modeling tasks have the emergent property of learning feature representations that correlate exceptionally well with human judgement. LPIPS uses this property to compare images by feeding both images to such a network, aggregating the output of intermediate layers $\mathbf{y}^l$ and finally computing their average euclidean distance:

$$\text{LPIPS} = \sum_{l}^{\text{layers}} \frac{1}{W_l H_l} \sum_{(i,j)}^{(W_l,H_l)} ||\mathbf{w}_l \odot (\mathbf{y}^l_{i,j} - \hat{\mathbf{y}}^l_{i,j})||_2^2 \tag{27}$$

Here $\mathbf{y}^l$ and $\hat{\mathbf{y}}^l$ are the activations of the network at layer l for I^{gt} and I, respectively. The dot product $\odot$ associated with weight $\mathbf{w}_l$ is a channel-wise weighted sum, in which the weights $\mathbf{w}_l$ are optimized by fine-tuning the network with a small network that predicts perceptual judgement based on a human-annotated dataset. In this thesis, we use an open source implementation of LPIPS[6] [70] using a trained VGG-architecture [71] as the base network.

6.2. Comparison

The comparison of the trained NeRF, NeX and IBR-Net models was carried out by synthesizing views using the camera poses that correspond to each validation image and then averaging the perceptual metrics for each scene and method. In addition to perceptual metrics, we provide qualitative comparison for each scene and method.

In Tables 5, 6, and 7, we see the results for the forward-facing scenes (Classroom 1, Classroom 2 and Cafeteria) using NeRF, NeX and IBR-Net. The qualitative results are seen in Figures 19-24.

Method	PSNR ↑	SSIM ↑	LPIPS ↓
NeRF	21.96	.7688	.5892
NeX	24.62	.8071	.5304
IBR-Net	**25.44**	**.8342**	**.4006**

Table 5. The average PSNR, SSIM, and LPIPS for the Classroom 1 -scene.

Method	PSNR ↑	SSIM ↑	LPIPS ↓
NeRF	**25.83**	**.8476**	.4871
NeX	24.37	.8231	.4808
IBR-Net	20.95	.8067	**.4291**

Table 6. The average PSNR, SSIM, and LPIPS for the Classroom 2 -scene.

[6] https://github.com/alexlee-gk/lpips-tensorflow

Method	PSNR ↑	SSIM ↑	LPIPS ↓
NeRF	**23.34**	.7482	.4406
NeX	23.30	**.8016**	.3730
IBR-Net	20.71	.7541	**.3247**

Table 7. The average PSNR, SSIM, and LPIPS for the Cafeteria -scene.

Similarly, the result for inward-facing scenes (Printer, Bike, Vase) using NeRF and IBR-Net are seen in Tables 8, 9 and 10. The qualitative results are seen in Figures 25-29.

Method	PSNR ↑	SSIM ↑	LPIPS ↓
NeRF	25.87	.7148	.6100
IBR-Net	**26.85**	**.7970**	**.3611**

Table 8. The average PSNR, SSIM, and LPIPS for the Printer -scene.

Method	PSNR ↑	SSIM ↑	LPIPS ↓
NeRF	20.53	.5991	.5612
IBR-Net	**20.70**	**.6872**	**.3586**

Table 9. The average PSNR, SSIM, and LPIPS for the Bike -scene.

Method	PSNR ↑	SSIM ↑	LPIPS ↓
NeRF	**26.74**	**.8478**	.4643
IBR-Net	23.49	.8414	**.4589**

Table 10. The average PSNR, SSIM, and LPIPS for the Vase -scene.

Finally, the rendering time of each method was measured and compared in Table 11. The rendering time did not change significantly across scenes, which is why the time is reported for the Classroom 1 scene only. For NeX, we additionally measured the rendering time for the cached MPI in their WebGL viewer using NVIDIA RTX 2080.

Method	Render time [seconds]
NeRF	116.0
NeX	15.39
NeX (WebGL)	**0.016**
IBR-Net	145.0

Table 11. Average rendering time for the Classroom 1 scene.

Figure 26. Qualitative comparison for the Classroom 1 scene. From left to right: Ground truth, NeRF, NeX, IBR-Net.

Figure 27. Zoom-in at the projector in the Classroom 1 scene. From left to right: Ground truth, NeRF, NeX, IBR-Net.

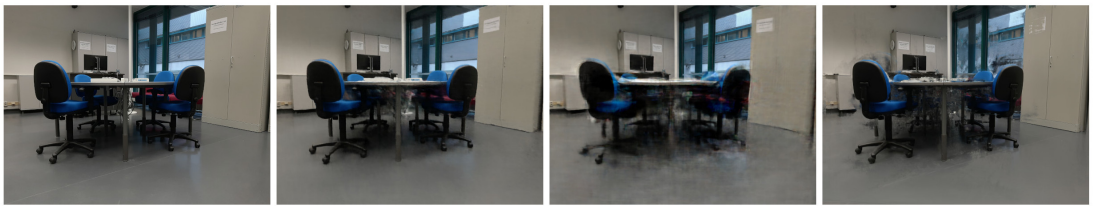


Figure 28. Qualitative comparison for the Classroom 2 scene. From left to right: Ground truth, NeRF, NeX, IBR-Net.

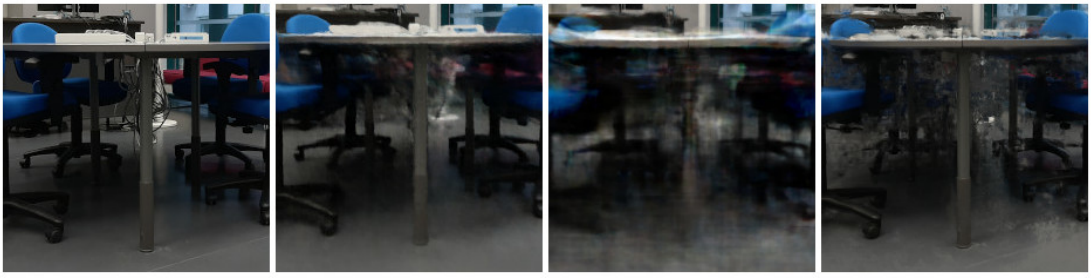


Figure 29. Zoom-in below the table in the Classroom 2 scene. From left to right: Ground truth, NeRF, NeX, IBR-Net.

Figure 30. Qualitative comparison for the Cafeteria scene. From left to right: Ground truth, NeRF, NeX, IBR-Net.

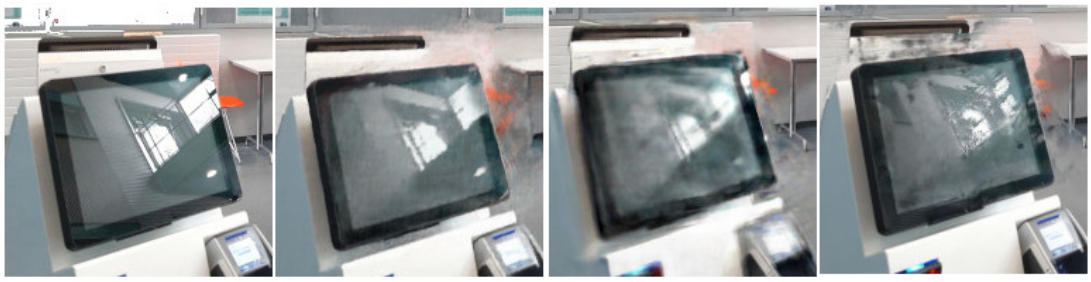


Figure 31. Zoom-in at the reflection in the Cafeteria scene. From left to right: Ground truth, NeRF, NeX, IBR-Net.

Figure 32. Qualitative comparison for the Printer scene. From left to right: Ground truth, NeRF, IBR-Net.

Figure 33. Zoom-in near the paper tray in the Printer scene. From left to right: Ground truth, NeRF, IBR-Net.

Figure 34. Qualitative comparison for the Bike scene. From left to right: Ground truth, NeRF , IBR-Net.

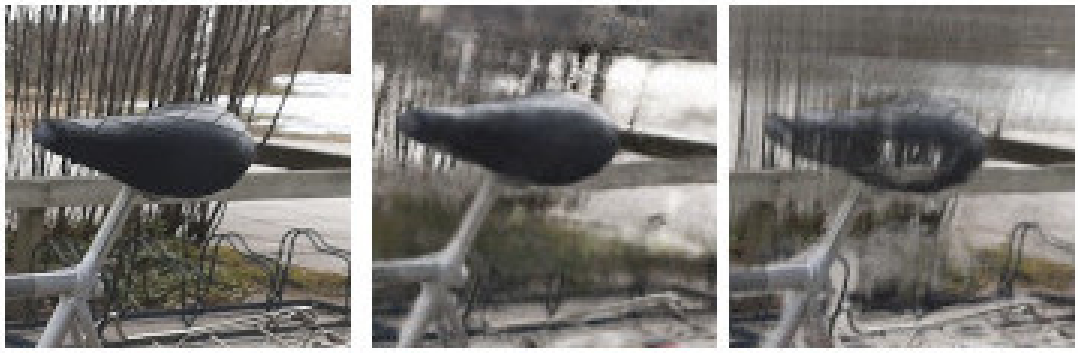


Figure 35. Zoom-in at the bench in the Bike scene. From left to right: Ground truth, NeRF, IBR-Net.

Figure 36. Qualitative comparison for the Vase scene. From left to right: Ground truth, NeRF, IBR-Net.

Figure 37. Zoom-in at the vase in the Vase scene. From left to right: Ground truth, NeRF, IBR-Net.

6.3. Additional Experiments

Next, we present the evaluation of additional experiments carried out with NeRF, NeRF−−, NeX and IBR-Net.

Inverse depth sampling experiment: To explore the effect of different sampling strategies, the Bike scene was trained an additional time using NeRF without inverse depth sampling, in which samples are drawn linearly in disparity rather than depth. The results are seen in Table 12 and the qualitative results in Figures 38 and 39.

Method	PSNR ↑	SSIM ↑	LPIPS ↓
NeRF (Regular sampling)	20.41	.5844	.5927
NeRF (Inverse depth sampling)	**20.53**	**.5991**	**.5612**

Table 12. Inverse depth sampling experiment: The average PSNR, SSIM, and LPIPS for the Bike -scene.

Calibration experiment: To explore the practical advantage of manual camera calibration, the Classroom 2 scene was trained with NeRF additionally without providing the estimated camera intrinsics (Table 2) to COLMAP, resulting in slightly different camera poses. The results for this experiment are seen in Table 13 and the qualitative results in Figure 40.

Method	PSNR ↑	SSIM ↑	LPIPS ↓
NeRF (with calibration)	**25.83**	**.8476**	**.4871**
NeRF (without calibration)	23.65	.7868	.5781

Table 13. Calibration experiment: The average PSNR, SSIM, and LPIPS for the Classroom 2 -scene.

NeRF−−: The evaluation process for NeRF−− was less straightforward. The camera poses that NeRF−− learns during the training process lie in a different space than the poses obtained from COLMAP and hence the pose of the evaluated image cannot be directly rendered. The workaround for this was to fix the weights of the network and optimize for the pose that minimizes MSE with respect to the validation image. The validation pose was searched for 1000 epochs for each validation image, then the corresponding view was rendered and evaluated against the validation image. The results are seen in Table 14 and the qualitative results are seen in Figure 41.

Method	PSNR ↑	SSIM ↑	LPIPS ↓
NeRF−−, Identity	13.05	.5746	.7398
NeRF−−, COLMAP	14.15	.6262	.6734
NeRF−−, Refined COLMAP	**14.16**	**.6281**	**.6727**

Table 14. NeRF−−: The average PSNR, SSIM, and LPIPS for the Classroom 2 -scene in three scenarios: Poses initialized from identity, fixed poses obtained from COLMAP and refined poses obtained from COLMAP.

Number of images experiment: Finally, the Cafeteria scene was evaluated additionally with IBR-Net using varying amount of aggregated source views (2, 4,

6, 8, 10) to explore how this affects the quality of synthesized views. The results of this experiment are seen in Table 15 and the qualitative results in Figures 42 and 43.

Num. images	PSNR ↑	SSIM ↑	LPIPS ↓
2	14.95	.5876	.4766
4	19.29	.7126	.3635
6	20.41	.7442	.3341
8	20.44	.7502	.3284
10	**20.71**	**.7541**	**.3247**

Table 15. Number of images experiment. The average PSNR, SSIM, and LPIPS for the Cafeteria scene using IBR-Net with varying number of aggregated source views.

Figure 38. Inverse depth sampling experiment. From left to right: Ground truth, NeRF with linear sampling, NeRF with inverse depth sampling.

Figure 39. Inverse depth sampling experiment. Zoom-in at the pedal in the Bike scene. From left to right: Ground truth, NeRF with linear sampling, NeRF with inverse depth sampling.

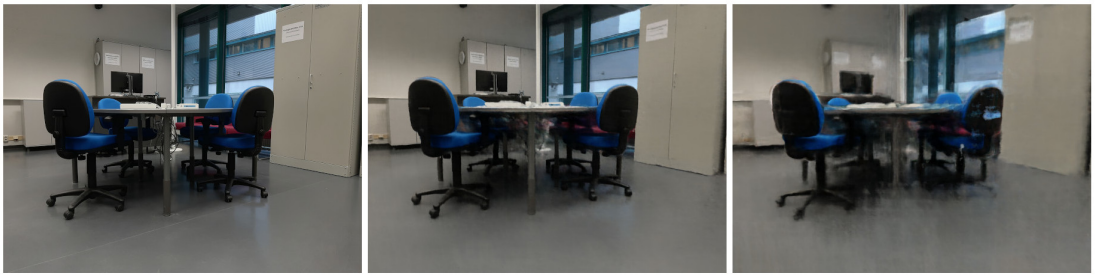


Figure 40. Calibration experiment: From left to right: Ground truth, NeRF with poses obtained by providing COLMAP with prior intrinsics, NeRF with poses obtained without providing prior intrinsics to COLMAP.

Figure 41. NeRF−−. From left to right: Ground truth, NeRF−− with poses initialized from identity, NeRF−− with poses initialized from COLMAP, NeRF−− with fixed poses from COLMAP.

Figure 42. Number of images experiment. From left to right: Ground truth, IBR-Net (2), IBR-Net (4), IBR-Net (6), IBR-Net (8) , IBR-Net (10).

Figure 43. Number of images experiment. Zoom in at the reflection. From left to right: Ground truth, IBR-Net (2), IBR-Net (4), IBR-Net (6), IBR-Net (8), IBR-Net (10).

7. DISCUSSION

In this chapter, we discuss the advantages, disadvantages and implications of different techniques in learning-based view synthesis based on the findings in this thesis. We discuss the results and limitations of the conducted experiments and finally consider the implications and future research directions.

7.1. Results of the Comparison

In the experimentation part of this thesis, we collected a variety of multi-view stereo datasets for the purposes of view synthesis and experimented with state of the art learning-based view synthesis algorithms, namely, NeRF, NeRF−−, NeX and IBR-Net. Overall, the quantitative and qualitative results show that each of the methods are capable of synthesizing plausible views from novel viewpoints, including situations with complex geometry as well as transparent (Figure 29) and reflective materials (Figure 23).

First, we address the overall results of the perceptual comparison. In terms of LPIPS, IBR-Net synthesized the most plausible views apart from a few outliers. However, in terms of PSNR, NeRF and NeX performed better in some scenarios. Looking at the qualitative results partly explains the conflicting results of the PSNR and LPIPS metrics, when considering the fact that PSNR places much less emphasis on blur when differentiating between images. IBR-Net clearly excels at preserving sharp details, while NeRF and NeX does not preserve sharp details as well, producing blurry views across all scenes. Increasing the number of iterations should however, increase the capability of NeRF and NeX to synthesize sharper views. Especially in dark and ambiguous regions, NeRF and NeX tend to degrade in a way that resembles blur. On the other hand, IBR-Net tends to generate spurious texture and anomalies instead (e.g. in Figure 30). In addition, when IBR-Net is faced in a situation where none of the aggregated images reach a region of the synthesized view, the region is rendered as black (Figure 22), whereas NeRF and NeX degrade somewhat gracefully.

The measured rendering times (Table 12) show that NeX is suited for real-time rendering when using the cached representation in WebGL. When using the network to synthesize views, NeX is also the fastest. As for NeRF and IBR-Net, the latter takes roughly 25% more time synthesize a novel view, which is understandable given the massive size of the network.

Moving on to the required amount of time to train a scene, IBR-Net is clearly most convenient given the pre-trained network. The pretrained network seems to translate very well to scenes outside of the training set, as was the case in these experiments. As for NeRF and NeX, these methods are not suited for pre-training. The latter converged more rapidly, taking just 1 hour to reach MSE < 0.05, whereas for NeRF it took roughly 10 hours to reach the same level (Figures 13 and 15).

The type of scene that NeRF and IBR-Net were compatible with are both forward-facing and inward-facing scenes, while NeX was limited to only forward-facing scenes due to the inherent limitations of the multi-plane image representation. The limitations of NeRF and IBR-Net in outdoor scenes is revealed in the Bike scene, where both methods struggle to reproduce the background content (Figure 33).

Speculating about the scalability of each method, NeX can easily run into memory limitations for larger scenes, as the resolution of the multi-plane images need to be increased as well. Both NeX and NeRF are limited in terms of signal complexity due to the fact that a fixed-size network is being used to implicitly represent the whole scene. IBR-Net is a promising approach for arbitrarily large scenes because it only uses information around a local region of the scene to synthesize a view.

7.2. Results of the Additional Experiments

Next, we discuss the results of the additional experiments presented in Section 6.3.

Inverse depth sampling experiment: The result of the inverse depth sampling experiment (Table 10, Figure 27) highlights the importance of choosing an appropriate sampling strategy for volume rendering. All of the metrics are improved when using inverse depth sampling and the qualitative results show that using linear sampling can miss important details near the camera as more samples are allocated near the outskirts of the scene.

Calibration experiment: The results of the calibration experiment (Table 11, Figure 30) indicate that using camera poses obtained by providing manually calibrated intrinsics to COLMAP, as opposed to letting COLMAP estimate them automatically, causes a significant improvement on the synthesized views. This is unfortunate for applications where manual calibration is not available and highlights the importance of using alternative methods to obtain accurate intrinsics in this situation. One potential solution is the approach of NeRF$--$, which can be used to refine prior estimates of the camera parameters.

NeRF$--$: The experiments carried out with NeRF$--$ (Table 14, Figure 40) show that refining the camera poses obtained from e.g. COLMAP can synthesize slightly better views compared to the situation where the camera parameters are fixed. This can especially be useful in scenarios where the camera parameters are not entirely accurate. Initializing the camera poses from identity seems to be a challenging task for NeRF$--$ and having some prior estimate greatly improves the results. The weak quantitative scores are explained by the validation pose search, in which the network is optimized to find the pose that minimizes MSE with respect to the validation image. Looking at the qualitative results, we see that the camera poses that were found are slightly offset from the true camera pose, which affects the metrics significantly. Increasing the amount of iterations should however, alleviate this issue.

Number of source views experiment: The experiment carried out with IBR-Net, in which the number of source views is varied (Table 15, Figures 42-43) show that IBR-Net can synthesize a plausible view with even just a few images. Increasing the number of source views from 6 resulted in diminishing returns, improving only slightly.

7.3. Limitations of the Study

The experiments conducted in this thesis were limited in some aspects. Firstly, the required time to train the applied networks was rather long and computationally demanding, which in turn limited the scale and amount of scenes to experiment with in

the time frame of this thesis. Second, all of the collected scenes were relatively small-scale ranging from 36-100 images. Third, when evaluating the methods there was no consideration of target poses that deviate far away from the scene. This is because each validation image had at least one nearby image in the training set. Finally, some of the methodologies discussed in the earlier part of this thesis were left out of the experimentation. Specifically, early methods in neural image-based rendering, neural re-rendering and other volumetric representations such as multi-sphere images and voxel grids.

7.4. Implications and Future Research Directions

Starting from coordinate based neural representations, we have seen that using radiance fields with volumetric rendering is an effective way to perform learning based view synthesis on scenes that are derived from image datasets. Compared to other geometrical representations such as 3D meshes and point clouds, neural radiance fields have become a promising way to store and synthesize views. As we have discussed in Section 3.4.2, an accelerating amount of research has been devoted to extending the scope of neural radiance fields, including relighting, scene composition, dynamic scenes and camera estimation. On the other hand, coordinate based neural representations are still time-consuming to train, and by themselves are slow to evaluate.

The slow rendering time of the above discussed networks can be solved by factorizing the network into position and view dependent components and then storing it to a volumetric data structure, e.g. multi-plane image as we have seen in NeX. In addition to multi-plane images, a number of alternatives have been proposed for other types of scenes, such as multi-plane spheres or octrees. An important research direction is how to store large-scale scenes in these data structures and how to update them in real-time for dynamic content. However, it is still possible that conventional 3D models will see use as a format for real-time rendering in novel view synthesis. The view synthesis algorithms used in this thesis can reliably estimate the geometry of the scene, which can be turned into a 3D model with e.g. the marching cubes algorithm [72]. Another research direction is how to encode view-dependent lighting and challenging geometry to such 3D models.

The arduous process of training a neural radiance field for each scene is partly solved by image-based rendering networks that aggregate nearby images to synthesize a local radiance field, as we have seen with IBR-Net. This approach is suited for pre-training and in addition, the network can focus on relevant parts of the scene to synthesize more detailed views. Combining volumetric data structures with image-based rendering networks would be a step towards applications that can rapidly construct and render radiance fields.

None of the experimented methods perform particularly well when faced with missing information, especially in situations where the target view contains scene content that was not seen during training. An important research direction is how to predict scene content in these missing regions, similar to generative adversarial networks that perform in-painting.

The use cases of learning-based novel view synthesis remains to be seen, but looking at current trends there are a number of directions the technology has potential to create innovation. Specifically in product visualization, the technology could let customers interact with products or assets in a fully interactive manner, for example by means of a virtual store or a virtual house tour. A similar application is virtual navigation, in which the user could explore a large-scale scene, such as a construction site or an office space. Aerial photography could be used for even larger scenes.

The advancements of current research in the field suggests that the technology could become a new medium for movies and video games. By using a multi-camera setup, the technology could be used to create interactive films where the viewer can freely move around and rewind the content. The technology could see some use in video game as well, by using assets derived from real-world images. However, the available software solutions that encapsulate the process of creating and rendering scenes with learning-based novel view synthesis is virtually non-existent. The creation of innovative software applications, akin to 3D modeling and photogrammetry software, is essential for widespread adoption of the technology.

8. CONCLUSION

This thesis has reviewed the concept of novel view synthesis, deep learning and surveyed novel view synthesis methodologies that utilize deep neural networks to synthesize views. We collected a novel multi-view stereo dataset for the purposes of view synthesis and demonstrated the process of collecting and preparing such datasets using a hand-held camera. The collected dataset contains a total of 6 scenes, roughly 40-100 images per scene, including different types of camera motion and scene content. The experimental part of this thesis adopted state of the art methods in learning-based novel view synthesis, namely, NeRF, NeRF−−, NeX and IBR-Net. The methods were compared quantitatively and qualitatively using the collected dataset and in addition, we experimented with some of their intrinsic properties to gain further insight on their performance.

The results show that each of the methods are capable of synthesizing plausible views from novel viewpoints, including situations with complex geometry as well as transparent and reflective materials. We have shown that using manually calibrated camera intrinsics resulted in significant improvement in synthesized views. The experiments done with NeRF−− reveal an interesting alternative to COLMAP for estimating camera poses from a set of images. The method could especially be useful when the camera parameters are not entirely accurate and need to be refined. However, compared to COLMAP the method is not very succesful at estimating camera poses without prior information. The results from NeX show that factorized neural representations are well suited for view synthesis and can be readily stored in a volumetric representation that enables fast rendering and portability to conventional graphics engines, such as WebGL. Finally, the results from IBR-Net show that using a pre-trained network that aggregates information from nearby views can replace the arduous process of training a neural radiance field for each scene, which is more scalable and greatly reduces the time to prepare a scene for view synthesis. Furthermore, IBR-Net excelled at preserving fine details of the scene even with just a few reference images, whereas NeRF and NeX was more prone to synthesize blurry views.

9. REFERENCES

[1] McMillan L. & Bishop G. (1995) Plenoptic modeling: An image-based rendering system. In: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, pp. 39–46.

[2] Levoy M. & Hanrahan P. (1996) Light field rendering. In: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, pp. 31–42.

[3] Gortler S.J., Grzeszczuk R., Szeliski R. & Cohen M.F. (1996) The lumigraph. In: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, pp. 43–54.

[4] Chai J.X., Tong X., Chan S.C. & Shum H.Y. (2000) Plenoptic sampling. In: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, pp. 307–318.

[5] Schonberger J.L. & Frahm J.M. (2016) Structure-from-motion revisited. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 4104–4113.

[6] Goesele M., Curless B. & Seitz S.M. (2006) Multi-view stereo revisited. In: 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), vol. 2, IEEE, vol. 2, pp. 2402–2409.

[7] Delaunay B., Vide S., Lamémoire A. & De Georges V. (1934) Bulletin de l'académie des sciences de l'urss. Classe des sciences mathématiques et naturelles 6, pp. 793–800.

[8] Chaurasia G., Duchene S., Sorkine-Hornung O. & Drettakis G. (2013) Depth synthesis and local warps for plausible image-based navigation. ACM Transactions on Graphics (TOG) 32, pp. 1–12.

[9] Goesele M., Ackermann J., Fuhrmann S., Haubold C., Klowsky R., Steedly D. & Szeliski R. (2010) Ambient point clouds for view interpolation. In: ACM SIGGRAPH 2010 papers, pp. 1–6.

[10] Hedman P., Ritschel T., Drettakis G. & Brostow G. (2016) Scalable inside-out image-based rendering. ACM Transactions on Graphics (TOG) 35, pp. 1–11.

[11] Penner E. & Zhang L. (2017) Soft 3d reconstruction for view synthesis. ACM Transactions on Graphics (TOG) 36, pp. 1–11.

[12] LeCun Y., Bengio Y. & Hinton G. (2015) Deep learning. nature 521, pp. 436–444.

[13] Duchi J., Hazan E. & Singer Y. (2011) Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research 12.

[14] Kingma D.P. & Ba J. (2014) Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 .

[15] Hecht-Nielsen R. (1992) Theory of the backpropagation neural network. In: Neural networks for perception, Elsevier, pp. 65–93.

[16] Ramchoun H., Idrissi M.A.J., Ghanou Y. & Ettaouil M. (2016) Multilayer perceptron: Architecture optimization and training. IJIMAI 4, pp. 26–30.

[17] Sitzmann V., Martel J., Bergman A., Lindell D. & Wetzstein G. (2020) Implicit neural representations with periodic activation functions. Advances in Neural Information Processing Systems 33.

[18] Mildenhall B., Srinivasan P.P., Tancik M., Barron J.T., Ramamoorthi R. & Ng R. (2020) Nerf: Representing scenes as neural radiance fields for view synthesis. In: European Conference on Computer Vision, Springer, pp. 405–421.

[19] O'Shea K. & Nash R. (2015) An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458 .

[20] He K., Zhang X., Ren S. & Sun J. (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778.

[21] Ronneberger O., Fischer P. & Brox T. (2015) U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical image computing and computer-assisted intervention, Springer, pp. 234–241.

[22] Gers F.A., Schmidhuber J. & Cummins F. (1999) Learning to forget: Continual prediction with lstm .

[23] Chung J., Gulcehre C., Cho K. & Bengio Y. (2015) Gated feedback recurrent neural networks. In: International conference on machine learning, PMLR, pp. 2067–2075.

[24] Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A.N., Kaiser L. & Polosukhin I. (2017) Attention is all you need. arXiv preprint arXiv:1706.03762 .

[25] Nguyen-Ha P., Huynh L., Rahtu E. & Heikkila J. (2020) Sequential view synthesis with transformer. In: Proceedings of the Asian Conference on Computer Vision.

[26] Wang Q., Wang Z., Genova K., Srinivasan P., Zhou H., Barron J.T., Martin-Brualla R., Snavely N. & Funkhouser T. (2021) Ibrnet: Learning multi-view image-based rendering. arXiv preprint arXiv:2102.13090 .

[27] Hedman P., Philip J., Price T., Frahm J.M., Drettakis G. & Brostow G. (2018) Deep blending for free-viewpoint image-based rendering. ACM Transactions on Graphics (TOG) 37, pp. 1–15.

[28] Riegler G. & Koltun V. (2020) Free view synthesis. In: European Conference on Computer Vision, Springer, pp. 623–640.

[29] Thies J., Zollhöfer M., Theobalt C., Stamminger M. & Nießner M. (2018) Ignor: Image-guided neural object rendering. arXiv preprint arXiv:1811.10720 .

[30] Choi I., Gallo O., Troccoli A., Kim M.H. & Kautz J. (2019) Extreme view synthesis. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 7781–7790.

[31] Nguyen P., Karnewar A., Huynh L., Rahtu E., Matas J. & Heikkila J. (2020) Rgbd-net: Predicting color and depth images for novel views synthesis. arXiv preprint arXiv:2011.14398 .

[32] Isola P., Zhu J.Y., Zhou T. & Efros A.A. (2017) Image-to-image translation with conditional adversarial networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1125–1134.

[33] Thies J., Zollhöfer M. & Nießner M. (2019) Deferred neural rendering: Image synthesis using neural textures. ACM Transactions on Graphics (TOG) 38, pp. 1–12.

[34] Meshry M., Goldman D.B., Khamis S., Hoppe H., Pandey R., Snavely N. & Martin-Brualla R. (2019) Neural rerendering in the wild. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 6878–6887.

[35] Aliev K.A., Ulyanov D. & Lempitsky V. (2019) Neural point-based graphics. arXiv preprint arXiv:1906.08240 2, p. 4.

[36] Zhou T., Tucker R., Flynn J., Fyffe G. & Snavely N. (2018) Stereo magnification: Learning view synthesis using multiplane images. arXiv preprint arXiv:1805.09817 .

[37] Mildenhall B., Srinivasan P.P., Ortiz-Cayon R., Kalantari N.K., Ramamoorthi R., Ng R. & Kar A. (2019) Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. ACM Transactions on Graphics (TOG) 38, pp. 1–14.

[38] Wizadwongsa S., Phongthawee P., Yenphraphai J. & Suwajanakorn S. (2021) Nex: Real-time view synthesis with neural basis expansion. arXiv preprint arXiv:2103.05606 .

[39] Attal B., Ling S., Gokaslan A., Richardt C. & Tompkin J. (2020) Matryodshka: Real-time 6dof video view synthesis using multi-sphere images. In: European Conference on Computer Vision, Springer, pp. 441–459.

[40] Hold-Geoffroy Y., DiVerdi S., Sun Q., Sunkavalli K. & Ramamoorthi R. Deep multi depth panoramas for view synthesis .

[41] Sitzmann V., Thies J., Heide F., Nießner M., Wetzstein G. & Zollhofer M. (2019) Deepvoxels: Learning persistent 3d feature embeddings. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2437–2446.

[42] Nguyen-Phuoc T., Li C., Theis L., Richardt C. & Yang Y.L. (2019) Hologan: Unsupervised learning of 3d representations from natural images. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 7588–7597.

[43] Nguyen-Phuoc T., Richardt C., Mai L., Yang Y.L. & Mitra N. (2020) Blockgan: Learning 3d object-aware scene representations from unlabelled images. arXiv preprint arXiv:2002.08988 .

[44] Lombardi S., Simon T., Saragih J., Schwartz G., Lehrmann A. & Sheikh Y. (2019) Neural volumes: Learning dynamic renderable volumes from images. arXiv preprint arXiv:1906.07751 .

[45] Laine S. & Karras T. (2010) Efficient sparse voxel octrees. IEEE Transactions on Visualization and Computer Graphics 17, pp. 1048–1059.

[46] Liu L., Gu J., Lin K.Z., Chua T.S. & Theobalt C. (2020) Neural sparse voxel fields. arXiv preprint arXiv:2007.11571 .

[47] Yu A., Li R., Tancik M., Li H., Ng R. & Kanazawa A. (2021) Plenoctrees for real-time rendering of neural radiance fields. arXiv preprint arXiv:2103.14024 .

[48] Mescheder L., Oechsle M., Niemeyer M., Nowozin S. & Geiger A. (2019) Occupancy networks: Learning 3d reconstruction in function space. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 4460–4470.

[49] Park J.J., Florence P., Straub J., Newcombe R. & Lovegrove S. (2019) Deepsdf: Learning continuous signed distance functions for shape representation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 165–174.

[50] Sitzmann V., Zollhöfer M. & Wetzstein G. (2019) Scene representation networks: Continuous 3d-structure-aware neural scene representations. arXiv preprint arXiv:1906.01618 .

[51] Niemeyer M., Mescheder L., Oechsle M. & Geiger A. (2020) Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 3504–3515.

[52] Tancik M., Srinivasan P.P., Mildenhall B., Fridovich-Keil S., Raghavan N., Singhal U., Ramamoorthi R., Barron J.T. & Ng R. (2020) Fourier features let networks learn high frequency functions in low dimensional domains. arXiv preprint arXiv:2006.10739 .

[53] Tancik M., Mildenhall B., Wang T., Schmidt D., Srinivasan P.P., Barron J.T. & Ng R. (2020) Learned initializations for optimizing coordinate-based neural representations. arXiv preprint arXiv:2012.02189 .

[54] Garbin S.J., Kowalski M., Johnson M., Shotton J. & Valentin J. (2021) Fastnerf: High-fidelity neural rendering at 200fps. arXiv preprint arXiv:2103.10380 .

[55] Kajiya J.T. & Von Herzen B.P. (1984) Ray tracing volume densities. ACM SIGGRAPH computer graphics 18, pp. 165–174.

[56] Martin-Brualla R., Radwan N., Sajjadi M.S., Barron J.T., Dosovitskiy A. & Duckworth D. (2020) Nerf in the wild: Neural radiance fields for unconstrained photo collections. arXiv preprint arXiv:2008.02268 .

[57] Zhang K., Riegler G., Snavely N. & Koltun V. (2020) Nerf++: Analyzing and improving neural radiance fields. arXiv preprint arXiv:2010.07492 .

[58] Yu A., Ye V., Tancik M. & Kanazawa A. (2020) pixelnerf: Neural radiance fields from one or few images. arXiv preprint arXiv:2012.02190 .

[59] Srinivasan P.P., Deng B., Zhang X., Tancik M., Mildenhall B. & Barron J.T. (2020) Nerv: Neural reflectance and visibility fields for relighting and view synthesis. arXiv preprint arXiv:2012.03927 .

[60] Kajiya J.T. (1986) The rendering equation. In: Proceedings of the 13th annual conference on Computer graphics and interactive techniques, pp. 143–150.

[61] Guo M., Fathi A., Wu J. & Funkhouser T. (2020) Object-centric neural scene rendering. arXiv preprint arXiv:2012.08503 .

[62] Xian W., Huang J.B., Kopf J. & Kim C. (2020) Space-time neural irradiance fields for free-viewpoint video. arXiv preprint arXiv:2011.12950 .

[63] Park K., Sinha U., Barron J.T., Bouaziz S., Goldman D.B., Seitz S.M. & Brualla R.M. (2020) Deformable neural radiance fields. arXiv preprint arXiv:2011.12948 .

[64] Wang Z., Wu S., Xie W., Chen M. & Prisacariu V.A. (2021) Nerf $--$: Neural radiance fields without known camera parameters. arXiv preprint arXiv:2102.07064 .

[65] Eslami S.A., Rezende D.J., Besse F., Viola F., Morcos A.S., Garnelo M., Ruderman A., Rusu A.A., Danihelka I., Gregor K. et al. (2018) Neural scene representation and rendering. Science 360, pp. 1204–1210.

[66] Opencv calibration tutorial. https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html.

[67] Wang Z., Bovik A.C., Sheikh H.R. & Simoncelli E.P. (2004) Image quality assessment: from error visibility to structural similarity. IEEE transactions on image processing 13, pp. 600–612.

[68] Zhang R., Isola P., Efros A.A., Shechtman E. & Wang O. (2018) The unreasonable effectiveness of deep features as a perceptual metric. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 586–595.

[69] Tensorflow ssim implementation. https://www.tensorflow.org/api_docs/python/tf/image/ssim.

[70] Tensorflow lpips implementation. https://github.com/alexlee-gk/lpips-tensorflow.

[71] Simonyan K. & Zisserman A. (2014) Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 .

[72] Lorensen W.E. & Cline H.E. (1987) Marching cubes: A high resolution 3d surface construction algorithm. ACM siggraph computer graphics 21, pp. 163–169.

10. APPENDIX

A. OpenCV Camera Calibration

The following Python script implements the calibration procedure discussed in Section 4.1.2. The output of this script are the estimated camera intrinsics. This implementation is largely based on the OpenCV calibration tutorial[7] [66].

```
import numpy as np
import cv2 as cv
import glob
import matplotlib.pyplot as plt

width = 11                  # Chessboard width
height = 7                  # Chessboard height
image_folder = "./images"   # Folder containing the calibration images
image_format = "jpg"        # Image format of the calibration images

objp = np.zeros((width*height,3), np.float32)
objp[:,:2] = np.mgrid[0:width,0:height].T.reshape(-1,2)
objpoints = []
imgpoints = []
images = glob.glob(image_folder + "/*." + image_format)

# Find chessboard corners and plot the results
for fname in images:
    img = cv.imread(fname)
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    ret, corners = cv.findChessboardCornersSB(gray, (width, height), None)
    if ret == True:
        objpoints.append(objp)
        criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)
        sp_corners = cv.cornerSubPix(gray, corners, (11,11), (-1,-1), criteria)
        imgpoints.append(sp_corners)
        plt.scatter(sp_corners.T[0], sp_corners.T[1], c="red", s=4.0)
        plt.imshow(gray, cmap="gray")
        plt.show()

# Calibrate the camera and print the results
ret, mtx, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints, gray.shape
    [::-1], None, None, flags=cv.CALIB_FIX_K3)
print("Camera matrix", mtx)
print("Distortion coefficients: ", dist)
print("RMS: ", ret)
```

B. Minimal NeRF implementation with PyTorch

The following program is a minimal PyTorch implementation of NeRF [18]. Note that the purpose of this program is not to exactly reproduce the original paper and it is missing many of the advanced features. This program serves as a practical demonstration of how to create and train a neural radiance field using PyTorch.

```
import torch
import numpy as np
import math
import time
import os
from torch import nn
from torch.utils.data import Dataset, DataLoader
from torch.nn import functional as F
from PIL import Image
```

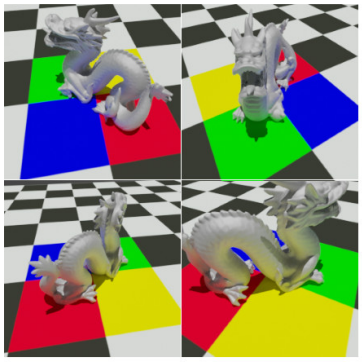
[7] https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html

```
class NeRF(nn.Module):

    def __init__(self, min_bounds, max_bounds, num_enc_p=10, num_enc_d=4,
        num_channels=256):
        super(NeRF, self).__init__()
        self.num_enc_p = num_enc_p
        self.num_enc_d = num_enc_d
        self.num_channels = num_channels
        self.min_bounds = min_bounds
        self.max_bounds = max_bounds
        self.layers = nn.ModuleList([
            self.layer(6 * num_enc_p, num_channels),
            self.layer(num_channels, num_channels),
            self.layer(num_channels, num_channels),
            self.layer(num_channels, num_channels),
            self.layer(num_channels, num_channels),
            self.layer(6 * num_enc_p + num_channels, num_channels),
            self.layer(num_channels, num_channels),
            self.layer(num_channels, num_channels),
            self.layer(num_channels, num_channels + 1, act_fn = torch.nn.Identity),
            self.layer(6 * num_enc_d + num_channels, num_channels // 2),
            self.layer(num_channels // 2, 3, act_fn = torch.nn.Sigmoid)
        ])

    def layer(self, in_features, out_features, act_fn = torch.nn.ReLU):
        return nn.Sequential(
            torch.nn.Linear(in_features, out_features),
            act_fn()
        )

    def get_rays(self, image, camera_pose, focal):
        W = image.shape[1]
        H = image.shape[0]
        i, j = torch.meshgrid(torch.linspace(0, W-1, W), torch.linspace(0, H-1, H))
        lat = (i - W/2) / W
        lon = (j - H/2) / H
        dirs = torch.stack([(i - (W - 1) * 0.5) / focal, -(j - (H - 1) * 0.5) / focal
            , -torch.ones_like(i)], -1)
        rays_d = torch.sum(dirs[..., np.newaxis, :] * camera_pose[:3,:3], -1)
        rays_d = rays_d.permute((1, 0, 2)) # (w, h, ch) -> (h, w, ch)
        rays_d = torch.reshape(rays_d, [-1,3])
        rays_d = rays_d / torch.sqrt(torch.sum(torch.square(rays_d), dim=1))[:,None]
        rays_o = camera_pose[:3,-1].expand(rays_d.shape)
        gt_colors = image.reshape([-1, 3])
        return [rays_o, rays_d, gt_colors]

    def box_intersection(self, positions, directions):
        inv_directions = 1 / directions
        t0 = (self.min_bounds - positions) * inv_directions
        t1 = (self.max_bounds - positions) * inv_directions
        tmax, _ = torch.min(torch.max(t0, t1), dim=1)
        return tmax

    def render_rays(self, positions, directions, num_samples):
        device = positions.device
        batch_size = positions.shape[0]
        path_length = self.box_intersection(positions, directions)
        samples = torch.arange(1, num_samples + 1).to(device) / num_samples
        p = positions[:,None,:] + directions[:,None,:] * samples[None,:,None] *
            path_length[:,None,None]
        p_flat = torch.reshape(p, (-1, 3)).float()
        d = directions.expand((num_samples, batch_size, 3)).permute((1, 0, 2))
        d_flat = torch.reshape(d, (-1, 3)).float()
        colors, densities = self.forward(p_flat, d_flat)
        colors = colors.reshape((batch_size, num_samples, 3))
        densities = densities.reshape(d.shape[:-1])
        delta = path_length / num_samples
        batch_ones = torch.ones((batch_size, 1)).to(device)
        alpha = 1.0 - torch.exp(-1.0 * densities * delta[:,None])
        T = torch.cumprod(torch.cat([batch_ones, 1.0 - alpha], -1), -1)[:, :-1]
        weights = T * alpha
```

```
        projected_colors = torch.sum(weights[:,:,None] * colors, dim=1)
        depth = torch.sum(weights * samples, dim=1)
        return [projected_colors, depth, weights]

    def encode(self, x, L):
        device = x.device
        batch_size = x.shape[0]
        f = ((2.0 ** torch.arange(0, L))).to(device)
        f = f.expand((batch_size, 3, L))
        f = torch.cat([torch.cos(math.pi * f * x[:,:,None]), torch.sin(math.pi * f *
            x[:,:,None])], dim=2)
        return f.reshape((batch_size, -1))

    def forward(self, p, d):
        p_normalized = -1. + 2. * (p - self.min_bounds) / (self.max_bounds - self.
            min_bounds)
        p_enc = self.encode(p_normalized, self.num_enc_p);
        d_enc = self.encode(d, self.num_enc_d);
        res1 = self.layers[0](p_enc)
        res2 = self.layers[1](res1)
        res3 = self.layers[2](res2)
        res4 = self.layers[3](res3)
        res5 = self.layers[4](res4)
        res6 = self.layers[5](torch.cat([p_enc, res5], dim=1))
        res7 = self.layers[6](res6)
        res8 = self.layers[7](res7)
        res9 = self.layers[8](res8)
        density = F.relu(res9[:,0])
        res10 = self.layers[9](torch.cat([res9[:,1:], d_enc], dim=1))
        color = self.layers[10](res10)
        return [color, density]

class MVSDataset(Dataset):

    def __init__(self, root_dir):
        self.root_dir = root_dir
        self.poses = torch.from_numpy(np.load(root_dir + "/poses.npy"))
        self.focal = 220.836477965

    def __len__(self):
        filenames = [f for f in os.listdir(self.root_dir) if (f[-3:] in ["jpg", "png
            "])]
        return len(filenames)

    def __getitem__(self, idx):
        if torch.is_tensor(idx): idx = idx.tolist()[0]
        img_path = self.root_dir + "/gt-%d.png" % (idx % self.__len__())
        image = Image.open(img_path) # (h, w, ch)
        image = torch.from_numpy(np.array(image)) / (256.0 - 1.0)
        pose = self.poses[idx]
        focal = self.focal
        return [image[:,:,:-1], pose, focal]

# Instantiate dataset & model
cuda = True
device = torch.device("cuda") if (cuda) else torch.device("cpu")
dataset = MVSDataset("./datasets/dragon")
min_bounds = torch.Tensor([-10, -10, -10]).to(device)
max_bounds = torch.Tensor([10, 10, 10]).to(device)
model = NeRF(min_bounds, max_bounds)
if (cuda): model.cuda()
total_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
print("Total model parameters: %d" % total_params)
print("Total images: %d" % len(dataset))
print("Image size: %s" % str(dataset[0][0].shape))
print("Focal length: %s" % str(dataset[0][2]))

# Training variables
optimizer = torch.optim.Adam(model.parameters(), lr=5e-4)
iterations = 1
rays_per_batch = 2**11
```

```
num_samples = 256
all_positions = []
all_directions = []
all_gt_colors = []

# Gather all rays
for i in range(len(dataset)):
    image, pose, focal = dataset[i]
    positions, directions, gt_colors = model.get_rays(image, pose, focal)
    all_positions.append(positions)
    all_directions.append(directions)
    all_gt_colors.append(gt_colors)

# Concatenate all rays
all_positions = torch.cat(all_positions, dim=0)
all_directions = torch.cat(all_directions, dim=0)
all_gt_colors = torch.cat(all_gt_colors, dim=0)

# Shuffle rays
shuffle = torch.randperm(all_positions.shape[0])
all_positions = all_positions[shuffle]
all_directions = all_directions[shuffle]
all_gt_colors = all_gt_colors[shuffle]

# Training loop
rays_per_iteration = all_positions.shape[0]
for i in range(iterations):
    current_idx = 0
    while(current_idx < rays_per_iteration):
        optimizer.zero_grad()
        indices = torch.arange(current_idx, min(all_positions.shape[0], current_idx +
             rays_per_batch))
        positions = all_positions[indices].to(device)
        directions = all_directions[indices].to(device)
        colors, depths, weights = model.render_rays(positions, directions,
            num_samples)
        gt = all_gt_colors[indices].to(device)
        loss = torch.mean(torch.square(colors - gt))
        loss.backward()
        current_idx += rays_per_batch
        optimizer.step()
        print('iteration: %d, loss: %.4f, ray count: %.2f%%' % (i, loss.item(), 100 *
             current_idx / rays_per_iteration))
    torch.save(model.state_dict(), "model-%d.pth" % i)
```

Synthetic Dataset (400 images)



Training

```
iteration: 0, loss: 0.0017, ray count: 99.86%
iteration: 0, loss: 0.0023, ray count: 99.87%
iteration: 0, loss: 0.0022, ray count: 99.88%
iteration: 0, loss: 0.0023, ray count: 99.88%
iteration: 0, loss: 0.0021, ray count: 99.89%
iteration: 0, loss: 0.0025, ray count: 99.90%
iteration: 0, loss: 0.0024, ray count: 99.91%
iteration: 0, loss: 0.0022, ray count: 99.91%
iteration: 0, loss: 0.0022, ray count: 99.92%
iteration: 0, loss: 0.0024, ray count: 99.93%
iteration: 0, loss: 0.0016, ray count: 99.94%
iteration: 0, loss: 0.0020, ray count: 99.95%
iteration: 0, loss: 0.0020, ray count: 99.95%
iteration: 0, loss: 0.0017, ray count: 99.96%
iteration: 0, loss: 0.0021, ray count: 99.97%
iteration: 0, loss: 0.0017, ray count: 99.98%
iteration: 0, loss: 0.0022, ray count: 99.98%
iteration: 0, loss: 0.0022, ray count: 99.99%
iteration: 0, loss: 0.0021, ray count: 100.00%
```

Synthesized Novel View

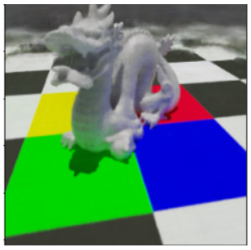


Figure 44. Results from the minimal NeRF program trained on a synthetic dataset.